

体系结构描述文档

灯具照明行业进销存管理系统



2014-11-3

小百合工作室

# 更新历史

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人员 | 日期 | 变更原因 | 版本号 |
| 陈云龙 | 2014-11-03 | 初步文档框架 | V0.0 |
| 陈云龙 | 2014-11-04 | 添加数据层分解 | V0.1 |
| 曾婧 | 2014-11-04 | 添加逻辑层初步分解 | V0.2 |
| 曾婧、陈云龙 | 2014-11-15 | 完成全部详细设计描述文档 | V1.0 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

更新历史 1

1. 引言 1

1.1. 编制目的 1

2. 产品概述 1

3. 体系结构设计概述 1

4. 结构视角 2

4.1. 业务逻辑层的分解 2

4.1.1. Commoditybl模块 2

4.1.2. CommoditySortbl模块 8

4.1.3. Inventorybl模块 13

4.1.4. Clientbl模块 19

4.1.5. Purchasebl模块 23

4.1.6. Salebl模块 28

4.1.7. Recordbl模块 34

4.1.8. Accountbl模块 38

4.1.9. AccountBillbl模块 41

4.1.10. CashBillbl模块 46

4.1.11. Accountainitbl模块 49

4.1.12. Approvalbl模块 52

4.1.13. Promotionbl模块 55

4.1.14. Userbl模块 60

4.2. 数据层的分解 65

4.2.1. DataFactoryService 65

4.2.2. InventoryDataService 68

4.2.3. PurchaseDataService 69

4.2.4. AccountBillDataService 70

4.2.5. ApprovalDataService 71

4.2.6. UserDataService 72

4.2.7. CommodityDataService 73

4.2.8. CommoditySortDataService 74

4.2.9. SaleDataService 75

4.2.10. AccountDataService 76

4.2.11. ClientDataService 77

4.2.12. AccountaInitDataService 78

4.2.13. PromotionDataService 79

5. 依赖视角 80

# 引言

## 编制目的

本报告详细完成灯具照明行业进销存系统的详细设计，达到指导后续软件构造的目的，同时实现和测试人员及用户的沟通。

本报告面向开发人员、测试人员及最终用户而编写，是了解系统的导航。

# 产品概述

参考灯具照明行业进销存系统用例文档和灯具照明行业进销存系统软件规格说明中对产品的概括描述。

# 体系结构设计概述

请参考进销存系统需求文档对体系结构设计的概述

# 结构视角

## 业务逻辑层的分解

业务逻辑层的开发包图详见体系结构文档图3。

### Commoditybl模块

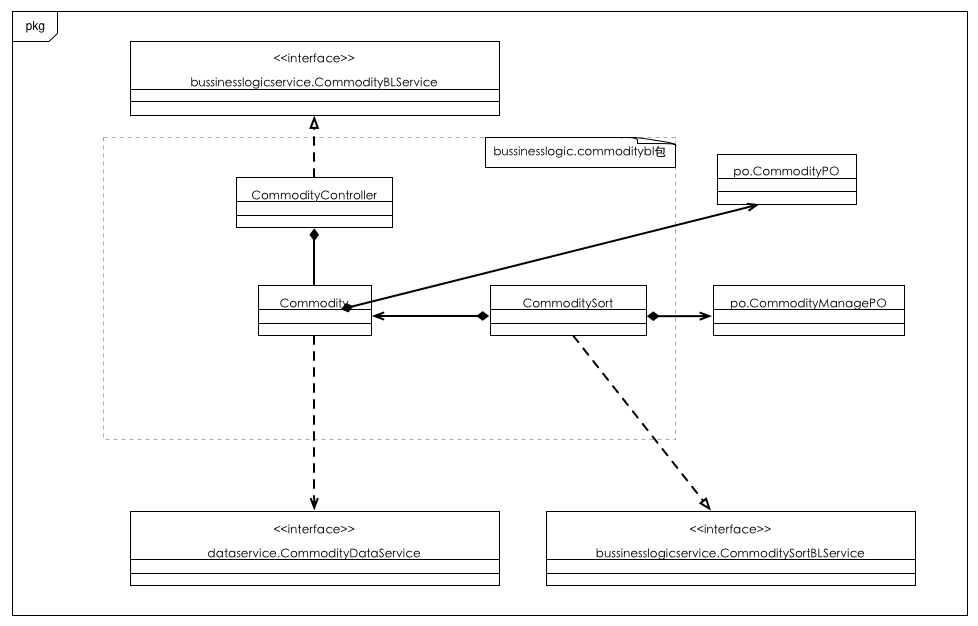
1. 模块概述

commoditybl模块负责实现商品分类和商品的管理。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.CommodityBLService接口。业务逻辑层和数据层之间添加dataservice.CommodityDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CommodityController，这样CommodityController会将对商品的逻辑业务处理委托给Commodity对象。CommodityPO是作为商品的持久化对象被添加到设计模型中去的。

Commoditybl模块的设计如图



commoditybl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| LoginController | 负责实现对应于登录界面所需要的服务 |
| CommodityController | 负责实现商品管理界面所需要的服务 |
| Commodity | 商品的领域模型对象，拥有商品的信息，可以帮助完成商品管理界面所需要的服务 |

1. 模块内部类的接口规范

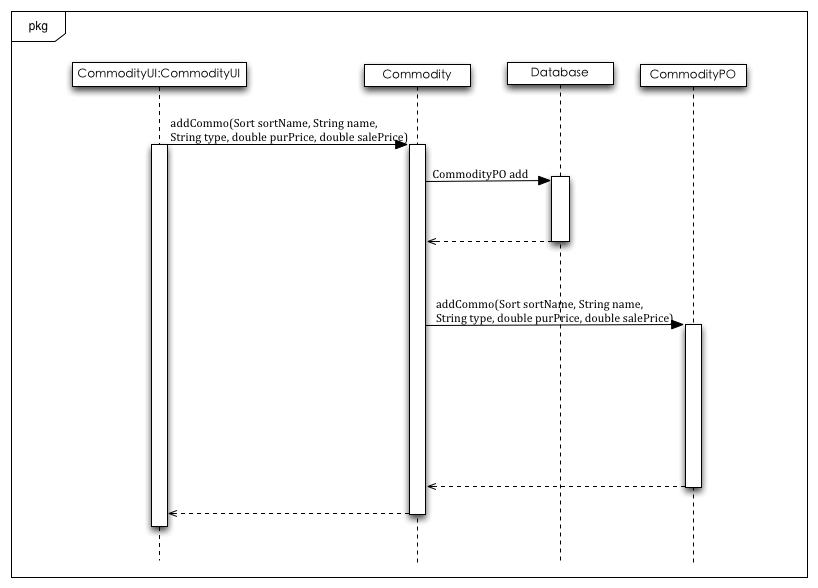
**CommodityController的接口规范**

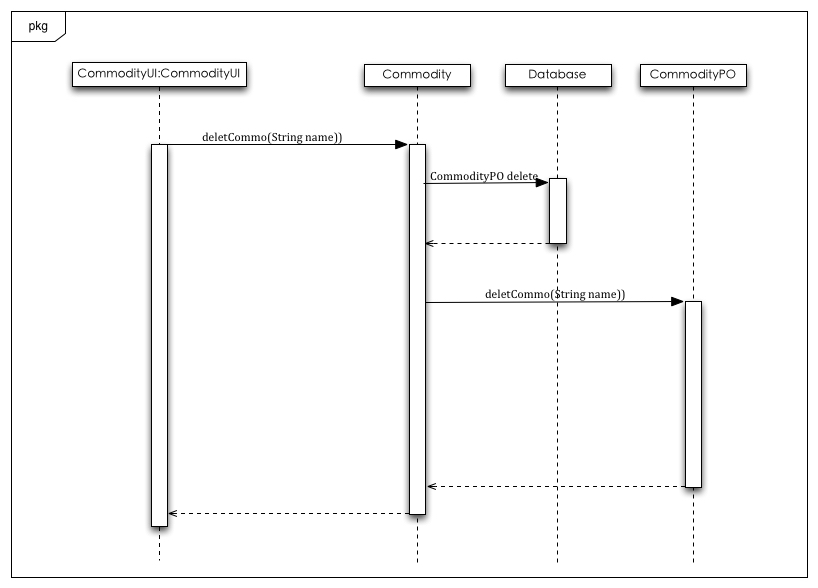
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommodityController.  addCommo | 语法 | public ResultMessage addCommo(Sort sortName, String name, String type, double purPrice, double salePrice) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 将商品添加到商品分类中，持久化更新涉及的对象的数据 |
| CommodityController.  deletCommo | 语法 | public ResultMessage deletCommo  (String name) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 删除该商品，持久化更新涉及的对象的数据 |
| CommodityController.  updCommo | 语法 | public ResultMessage updCommo  (String name, CommoditySort sortName, String type, double purPrice, double salePrice) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 商品更新，持久化更新涉及的对象的数据 |
| Commodity.findCommo | 语法 | public ResultMessage findCommo  (String name, String id, String type) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 查找到该商品 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Commodity.addCommo(Sort sortName, String name, String type, double purPrice, double salePrice) | 加入一个商品 | |
| Commodity.deletCommo  (String name) | 删除一个商品 | |
| Commodity.updCommo  (String name, CommoditySort sortName, String type, double purPrice, double salePrice) | 更改一个商品的信息 | |
| Commodity.findCommo  (String name, String id, String type) | 查找一个商品 | |

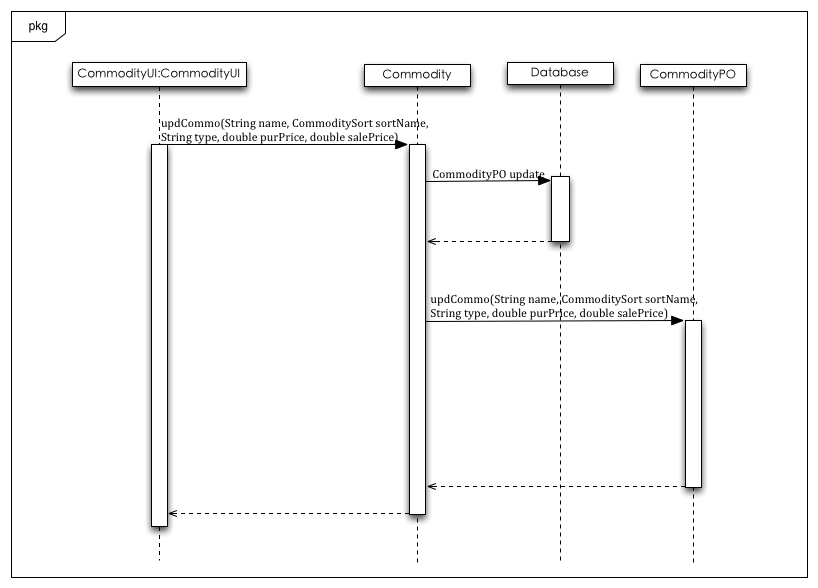
**Commodity的接口规范**

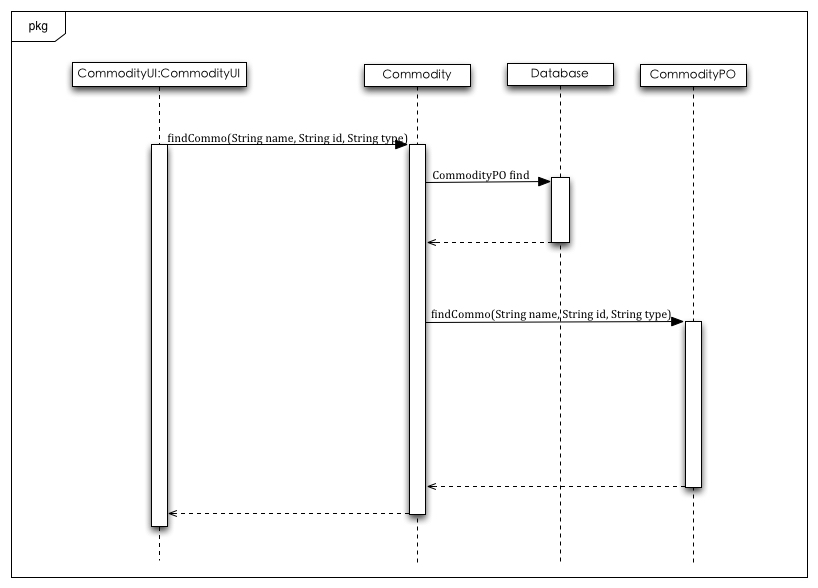
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Commodity.  addCommo | 语法 | public ResultMessage addCommo(Sort sortName, String name, String type, double purPrice, double salePrice) |
| 前置条件 | 分类、商品名称、型号、进价、零售价输入合法 |
| 后置条件 | 将商品添加到商品分类中，持久化更新涉及的对象的数据 |
| Commodity.  deletCommo | 语法 | public ResultMessage deletCommo  (String name) |
| 前置条件 | 商品名称输入合法，商品没有被操作过 |
| 后置条件 | 删除该商品，持久化更新涉及的对象的数据 |
| Commodity.  updCommo | 语法 | public ResultMessage updCommo  (String name, CommoditySort sortName, String type, double purPrice, double salePrice) |
| 前置条件 | 商品名称输入合法，更改信息合法 |
| 后置条件 | 商品更新，持久化更新涉及的对象的数据 |
| Commodity.  findCommo | 语法 | public ResultMessage findCommo  (String name, String id, String type) |
| 前置条件 | 商品名称输入合法，id输入合法 |
| 后置条件 | 查找到该商品 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| CommodityDataService.  findCommoSort(int id) | 根据ID进行查找单一持久化对象 | |
| CommodityDataService.  findCommo (int id) | 根据ID进行查找单一持久化对象 | |
| CommodityDataService.  findCommos  (Sting field, int id) | 根据字段名和值进行查找多个持久化对象 | |
| CommodityDataService.  add(CommodityPO po) | 添加单一化持久对象 | |
| CommodityDataService.  delete(CommodityPO po) | 删除单一化持久对象 | |
| CommodityDataService.  update(CommodityPO po) | 更改单一化持久对象 | |
| CommodityDataService.  getCommodityData | 得到Commodity数据的服务引用 | |

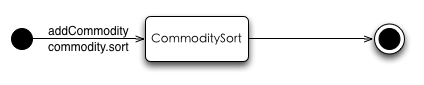
1. 业务逻辑层的动态模型











1. 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

### CommoditySortbl模块

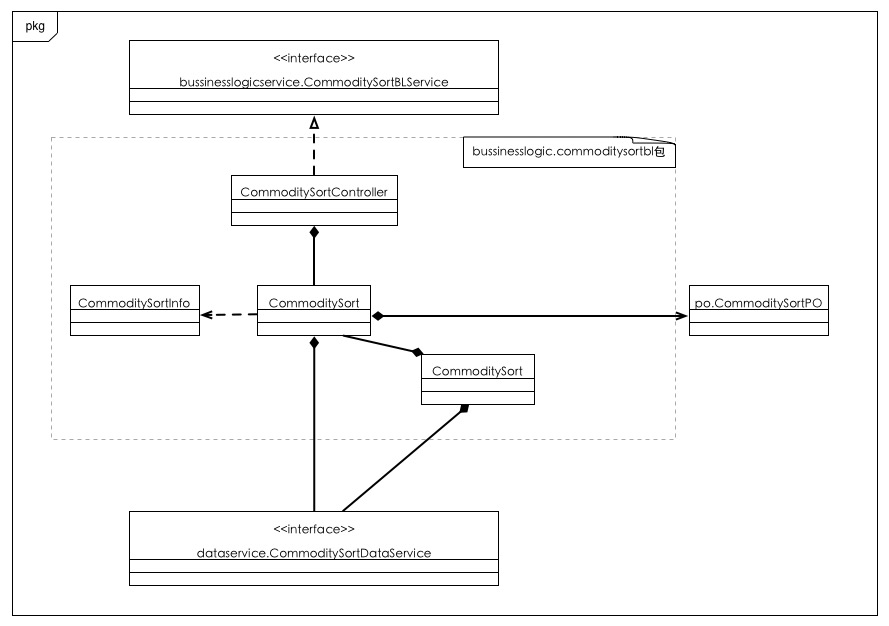
1. 模块概述

CommoditySortbl模块负责实现商品分类和商品的管理。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.CommoditySortBLService接口。业务逻辑层和数据层之间添加dataservice.CommoditySortDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CommoditySortController，这样CommoditySortController会将对商品分类的逻辑业务处理委托给CommoditySort对象。CommoditySortPO是作为商品类别的持久化对象被添加到设计模型中去的。

commoditysortbl模块的设计如图



commoditybl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| CommoditySortController | 负责实现商品分类管理界面所需要的服务 |
| CommoditySort | 商品分类的领域模型对象，拥有商品分类的信息，可以帮助完成商品分类界面所需要的服务 |
| CommodityInfo | 负责实现为其他包提供商品分类信息、更改商品分类信息的服务 |

1. 模块内部类的接口规范

CommoditySortController的接口规范如下表所示

**CommoditySortController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommoditySortController.  getID | 语法 | public String getID () |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Commodity |
| CommoditySortController.  addCommoSort | 语法 | public ResultMessage addCommoSort  (String sortName, String fatherID) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Commodity |
| CommoditySortController.  deletCommoSort | 语法 | public ResultMessage deleteCommoSort(  String ID) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Commodity，删除商品分类，持久化更新涉及的对象的数据 |
| CommoditySortController.  updCommoSort | 语法 | public ResultMessage updCommoSort  (Sring ID, String name) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合输入规则 |
| 后置条件 | 调用Commodity，更改商品分类，持久化更新涉及的对象的数据 |
| CommoditySortController.  show | 语法 | public ArrayList<CommoditySortVO> show() |
| 前置条件 | 已经创建一个Commodity领域对象 |
| 后置条件 | 调用Commodity，展示所有分类 |
| CommoditySortController.  show | 语法 | public ArrayList<CommoditySortVO>  show(String ID) |
| 前置条件 | 已经创建一个Commodity领域对象，并且输入符合规范 |
| 后置条件 | 调用Commodity，根据输入的ID展示分类 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| CommoditySort.  addCommoSort  (String sortName,  CommoditySort parentSort) | 加入一个商品分类 | |
| CommoditySort.  deleteCommoSort  (String ID) | 删除商品分类 | |
| CommoditySort.  updCommoSort  (Sring ID, String name) | 更改商品分类名称 | |
| CommoditySort.getID() | 得到新建分类的ID | |
| CommoditySort.show() | 展示所有分类 | |
| CommoditySort.show(String ID) | 根据ID展示分类 | |

CommoditySort的接口规范如下表所示

**Commoditysort的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommoditySort.  getID | 语法 | public String getID () |
| 前置条件 | 输入符合输入规则 |
| 后置条件 | 返回新建的ID |
| CommoditySort.  addCommoSort | 语法 | public ResultMessage addCommoSort  (String sortName, CommoditySort parentSort) |
| 前置条件 | 分类名称输入合法，并且父类中没有添加商品 |
| 后置条件 | 添加新的商品分类，持久化更对象的数据 |
| CommoditySort.  deletCommoSort | 语法 | public ResultMessage deleteCommoSort(String ID) |
| 前置条件 | 分类名称输入合法 |
| 后置条件 | 删除商品分类，持久化更新涉及的对象的数据 |
| CommoditySort.  updCommoSort | 语法 | public ResultMessage updCommoSort  (Sring ID, String name) |
| 前置条件 | 分类名称输入合法 |
| 后置条件 | 更改商品分类，持久化更新涉及的对象的数据 |
| CommoditySort.show | 语法 | public ArrayList<CommoditySortVO> show() |
| 前置条件 | 已经创建一个Commodity领域对象 |
| 后置条件 | 展示所有分类 |
| CommoditySort.show | 语法 | public ArrayList<CommoditySortVO>  show(String ID) |
| 前置条件 | ID输入合法 |
| 后置条件 | 根据输入的ID展示分类 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| CommoditySortDataService.  getID() | 得到新建分类的ID | |
| CommoditySortDataService.  find (int ID) | 根据ID进行查找单一持久化对象 | |
| CommoditySortDataService.  insert(CommoditySortPO po) | 添加单一化持久对象 | |
| CommoditySortDataService.  delete(CommoditySortPO po) | 删除单一化持久对象 | |
| CommoditySortDataService.  update(CommoditySortPO po) | 更改单一化持久对象 | |
| CommoditySortDataService.  getCommodityData | 得到CommoditySort数据的服务引用 | |

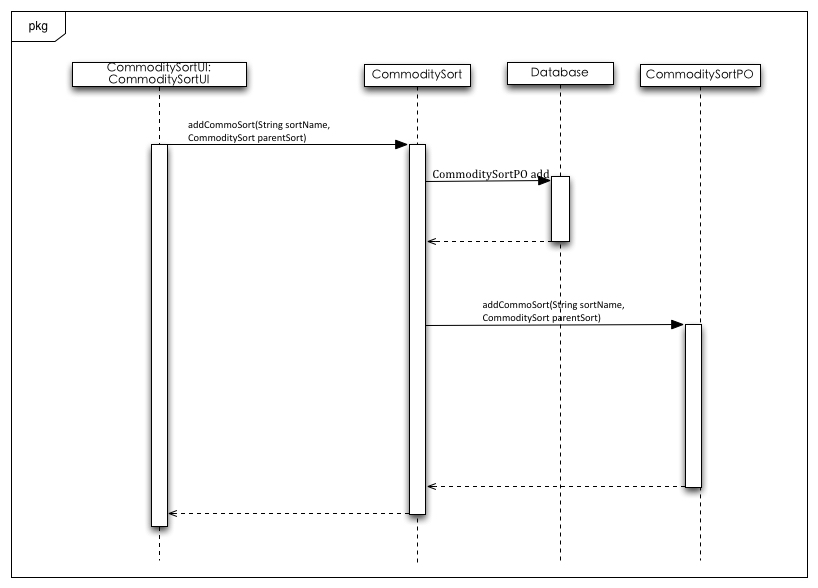
CommodityInfo的接口规范如下表所示

**CommodityInfo的接口规范**

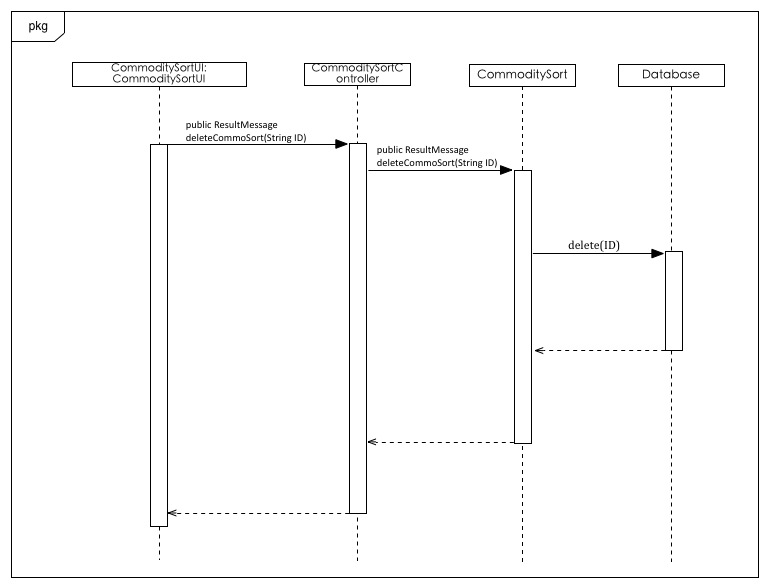
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommoditySortInfo.  getSortPOs | 语法 | public ArrayList<CommoditySortPO> getSortPOs() |
| 前置条件 | 其他类需要商品分类信息 |
| 后置条件 | 返回所有的分类的持久化对象 |
| CommoditySortInfo.  getSortVOs | 语法 | public ArrayList<CommoditySortVO>  getSortVOs(ArrayList<CommoditySortPO> POs) |
| 前置条件 | 其他类需要商品分类信息 |
| 后置条件 | 返回所有的分类的对象 |
| CommoditySortInfo.  deleteCommodity | 语法 | public void deleteCommodity  (String sortID, String ID) |
| 前置条件 | 该分类的商品可以删除 |
| 后置条件 | 删除商品，持久化更新涉及的对象的数据 |
| CommoditySortInfo.  addCommodity | 语法 | public void addCommodity(String sortID, String ID) |
| 前置条件 | 在分类中添加商品成功 |
| 后置条件 | 更改持久化更新涉及的对象的数据 |
| CommoditySortInfo. hasLeaf | 语法 | public boolean hasLeaf(String sortID) |
| 前置条件 | 需要对商品分类信息进行判断 |
| 后置条件 | 返回该商品分类是否有子类 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| CommoditySortDataService.  update(CommoditySortPO po) | 更改单一化持久对象 | |
| CommoditySort.  show() | 得到CommoditySort的所有持久化对象 | |

1. 业务逻辑层的动态模型

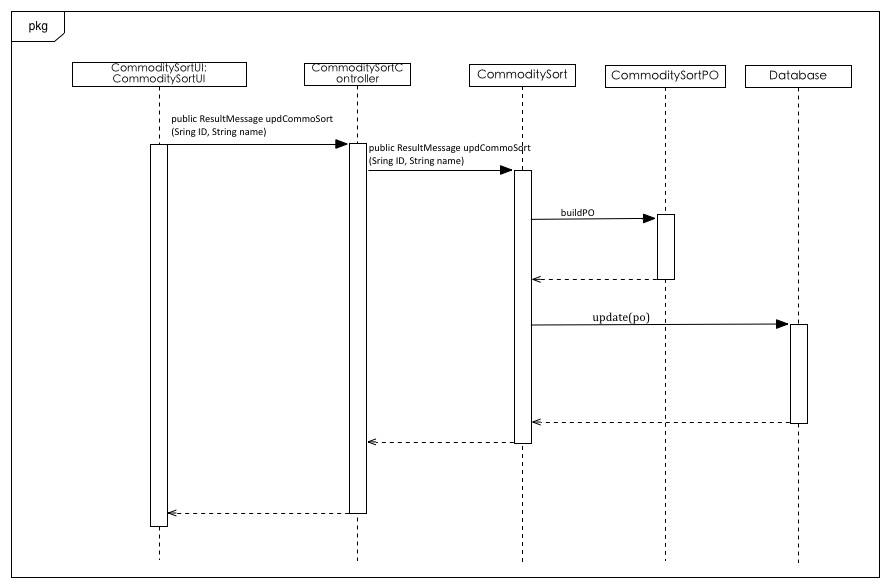
如下图表示了在进销存系统中，当库存管理人员想要添加一个商品分类的时候，库存管理业务逻辑处理的相关对象之间的协作



如下图表示了在进销存系统中，当库存管理人员想要添加一个商品分类的时候，库存管理业务逻辑处理的相关对象之间的协作



如下图表示了在进销存系统中，当库存管理人员想要更改一个商品分类的时候，库存管理业务逻辑处理的相关对象之间的协作

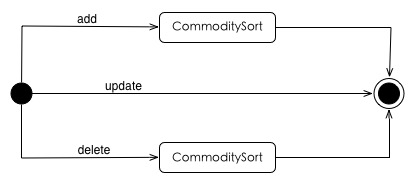


如下图所示的状态图描述了CommoditySort对象的生存期间的状态序列、引起转移的事件，以及因转移而伴随的动作。

随着deleteCommoditySort的方法被UI调用，CommoditySort进入判断是否可以删除的状态，如果判断其中没有商品，就可以将该分类删除，进入完成状态。

随着addCommoditySort的方法被UI调用，CommoditySort进入判断是否可以添加商品分类的状态，如果判断添加的父类中没有商品，则可以添加该分类，进入完成状态。

随着updateCommoditySort的方法被UI调用，可以修改该分类的名称，进入完成状态。



1. 业务逻辑层的设计原理

利用委托式控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象

### Inventorybl模块

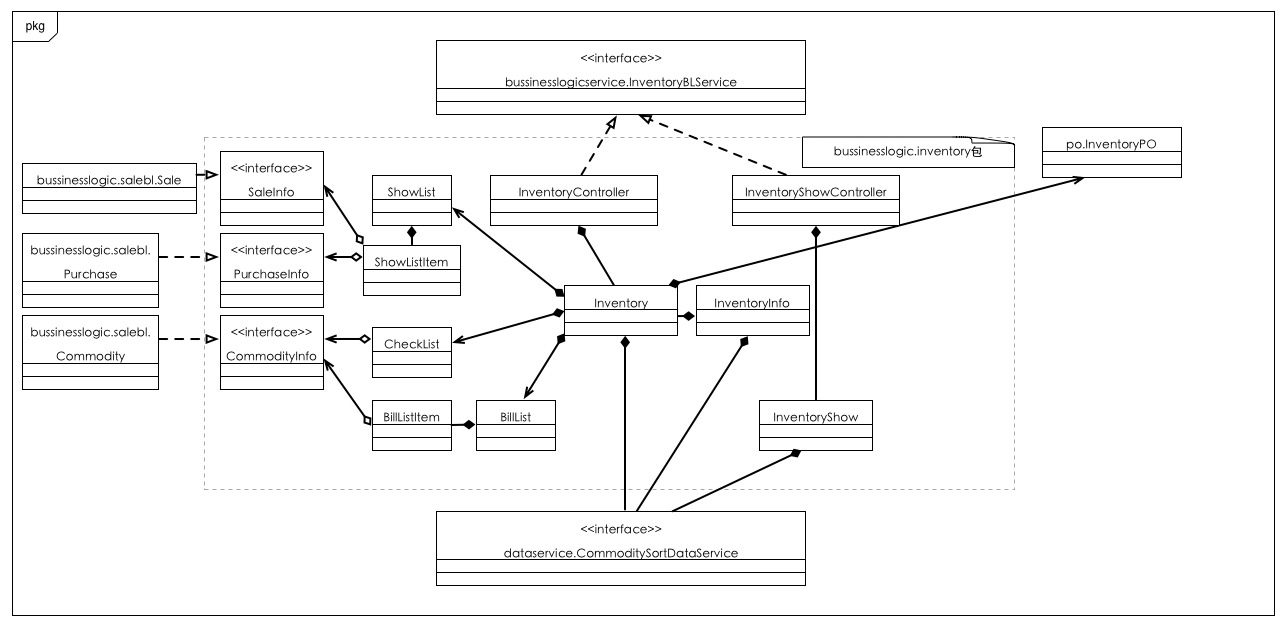
1. 模块概述

Inventorybl模块负责实现仓库的管理，对库存进行报损、报溢，建立赠送单，对库存进行盘点、查看。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. InventoryBLService接口。业务逻辑层和数据层之间添加dataservice. InventoryDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了InventoryController，这样InventoryController会将对库存的逻辑业务处理委托给Inventory对象。

Inventorybl模块的设计如图所示



**Inventortybl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| InventoryController | 负责实现仓库管理界面所需要的服务 |
| Inventory | 仓库管理的领域模型对象，拥有仓库的信息，可以帮助完成仓库管理界面所需要的服务 |
| InventoryShowController | 负责实现显示仓库单据所需要的服务 |
| InventoryShow | 仓库单据显示的领域模型对象，拥有仓库单据的信息 |
| BillList | 仓库赠送、报损、报溢、报警单的创建 |
| BillListItem | 仓库赠送、报损、报溢、报警单的条目内容 |
| InventoryInfo | 负责实现其他包对仓库的调用 |
| CheckList | 负责实现仓库的盘点 |
| CheckListItem | 仓库的盘点列表 |
| ViewList | 负责实现库存的查看 |

1. 模块内部类的接口规范

InventoryController的接口规范如下表

**InventoryController接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| InventoryController.  viewInventory | 语法 | public InventoryViewVO viewInventory  (Date beginDate, Date endDate) |
| 前置条件 | 已创建一个Inventory对象，并且输入合法 |
| 后置条件 | 调用Inventory的方法 |
| InventoryController.  checkRecord | 语法 | public InventoryCheckVO checkRecord( ) |
| 前置条件 | 已创建一个Inventory对象 |
| 后置条件 | 调用Inventory的方法 |
| InventoryController.  getGiftID | 语法 | public String getGiftID () |
| 前置条件 | 已创建一个Inventory对象 |
| 后置条件 | 调用Inventory的方法 |
| InventoryController.  getOverFlowID | 语法 | public String getOverFlowID () |
| 前置条件 | 已创建一个Inventory对象 |
| 后置条件 | 调用Inventory的方法 |
| Inventory.getLossID | 语法 | public String getLossID () |
| 前置条件 | 已创建一个Inventory对象 |
| 后置条件 | 调用Inventory的方法 |
| InventoryController.  getAlarmID | 语法 | public String getAlarmID () |
| 前置条件 | 已创建一个Inventory对象 |
| 后置条件 | 调用Inventory的方法 |
| InventoryController. addCommodity | 语法 | public void addCommodity(String ID, int number) |
| 前置条件 | 已创建一个Inventory对象，并且输入合法 |
| 后置条件 | 调用Inventory的方法 |
| InventoryController.  submit | 语法 | public InventoryBillVO submit(String remark) |
| 前置条件 | 已创建一个Inventory对象，并且输入合法 |
| 后置条件 | 调用Inventory的方法 |
| InventoryController. save | 语法 | public InventoryBillVO save(String remark) |
| 前置条件 | 已创建一个Inventory对象，并且输入合法 |
| 后置条件 | 调用Inventory的方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Inventory. viewInventory  (Date beginDate,  Date endDate) | 库存查看 | |
| Inventory.checkRecord( ) | 库存盘点 | |
| Inventory.getGiftID() | 得到赠送单ID | |
| Inventory.getOverFlowID | 得到报溢单ID | |
| Inventory.getLossID() | 得到报损单ID | |
| Inventory.getAlarmID() | 得到报警单ID | |
| Inventory.addCommodity  (String ID, int number) | 添加商品进单据 | |
| Inventory.submit(String r) | 提交单据 | |
| Inventory.save(String r) | 保存单据为草稿 | |

Inventory的接口规范如下表

**Inventory接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Inventory.viewInventory | 语法 | public InventoryViewVO viewInventory  (Date beginDate, Date endDate) |
| 前置条件 | 日期的输入合法 |
| 后置条件 | 显示在此时间段内的出／入库数量／金额，销售／进货数量／金额，以及库存合计 |
| Inventory.checkRecord | 语法 | public InventoryCheckVO checkRecord( ) |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 显示截至盘点时间的库存快照 |
| Inventory.getGiftID | 语法 | public String getGiftID () |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 返回对应的库存赠送单的ID |
| Inventory.getOverFlowID | 语法 | public String getOverFlowID () |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 返回对应的库存报溢单的ID |
| Inventory.getLossID | 语法 | public String getLossID () |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 返回对应的库存报损单的ID |
| Inventory.getAlarmID | 语法 | public String getAlarmID () |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 返回对应的库存警报单的ID |
| Inventory.addCommodity | 语法 | public void addCommodity(String ID, int number) |
| 前置条件 | 完成库存的盘点，输入数据 |
| 后置条件 | 生成相应的库存报溢单 |
| Inventory.submit | 语法 | public InventoryBillVO submit(String remark) |
| 前置条件 | 输入数据合法 |
| 后置条件 | 生成对应的单据 |
| Inventory.save | 语法 | public InventoryBillVO save(String remark) |
| 前置条件 | 输入的数据合法 |
| 后置条件 | 保存对应的单据为草稿 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| InventoryDataService.  insert(InventoryBillPO po) | 生成单据，告知是否成功 | |
| InventoryDataService.  update(InventoryBillPO po) | 更改单据数据 | |

InventoryShowController的接口规范如下表

**InventoryShowController接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| InventoryShowController.  showGifts | 语法 | public ArrayList<InventoryBillVO> showGifts() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  showOverFlow | 语法 | public ArrayList<InventoryBillVO>  showOverFlow() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  getGiftID | 语法 | public ArrayList<InventoryBillVO> showLoss() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  getOverFlowID | 语法 | public ArrayList<InventoryBillVO> showAlarm() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  getLossID | 语法 | public ArrayList<InventoryBillVO>  showGiftsApproving() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  getAlarmID | 语法 | public ArrayList<InventoryBillVO>  showOverFlowApproving() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController. addCommodity | 语法 | public ArrayList<InventoryBillVO>  showLossApproving() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  submit | 语法 | public ArrayList<InventoryBillVO>  showAlarmApproving() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController. save | 语法 | public ArrayList<InventoryBillVO>  showGiftsPass() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  viewInventory | 语法 | public ArrayList<InventoryBillVO>  showOverFlowPass() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  checkRecord | 语法 | public ArrayList<InventoryBillVO>  showLossPass() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  getGiftID | 语法 | public ArrayList<InventoryBillVO>  showAlarmPass() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  getOverFlowID | 语法 | public ArrayList<InventoryBillVO>  showGiftsFailure() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  getLossID | 语法 | public ArrayList<InventoryBillVO>  showOverFlowFailure() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  getAlarmID | 语法 | public ArrayList<InventoryBillVO>  showLossFailure() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController. addCommodity | 语法 | public ArrayList<InventoryBillVO>  showAlarmFailure() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  submit | 语法 | public ArrayList<InventoryBillVO>  showGiftsDraft() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController. save | 语法 | public ArrayList<InventoryBillVO>  showOverFlowDraft() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController.  submit | 语法 | public ArrayList<InventoryBillVO>  showLossDraft() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| InventoryShowController. save | 语法 | public ArrayList<InventoryBillVO>  showAlarmDraft() |
| 前置条件 | 已创建一个InventoryShow对象 |
| 后置条件 | 调用InventoryShow的方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Inventory. viewInventory  (Date beginDate,  Date endDate) | 库存查看 | |
| Inventory.checkRecord( ) | 库存盘点 | |
| Inventory.getGiftID() | 得到赠送单ID | |
| Inventory.getOverFlowID | 得到报溢单ID | |
| Inventory.getLossID() | 得到报损单ID | |
| Inventory.getAlarmID() | 得到报警单ID | |
| Inventory.addCommodity  (String ID, int number) | 添加商品进单据 | |
| Inventory.submit(String r) | 提交单据 | |
| Inventory.save(String r) | 保存单据为草稿 | |

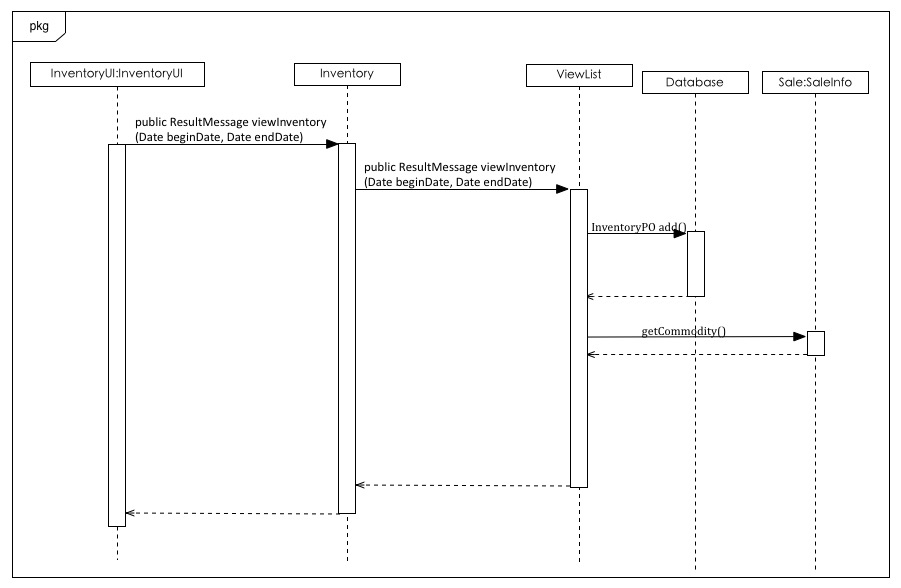
InventoryShow的接口规范如下表

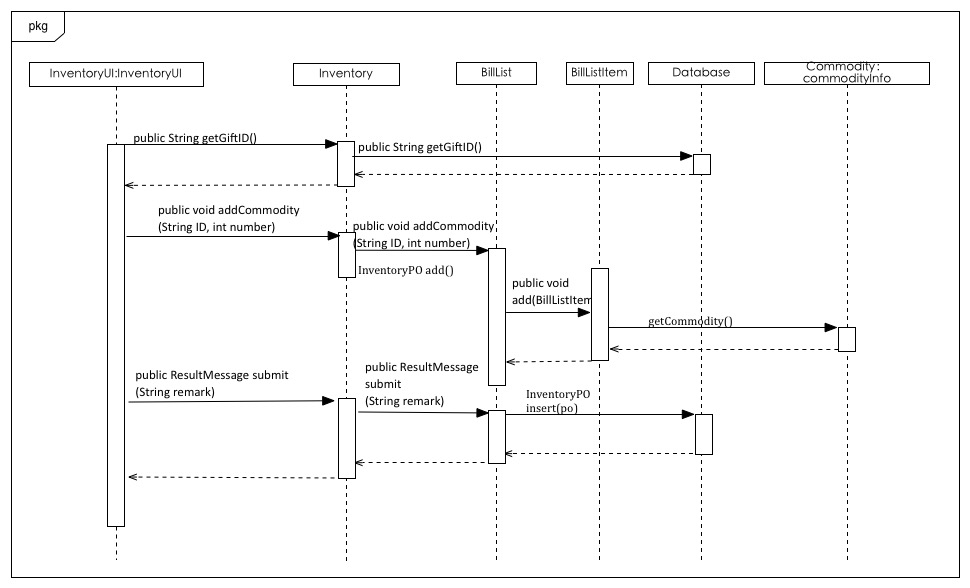
**InventoryShow接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Inventory.viewInventory | 语法 | public InventoryViewVO viewInventory  (Date beginDate, Date endDate) |
| 前置条件 | 日期的输入合法 |
| 后置条件 | 显示在此时间段内的出／入库数量／金额，销售／进货数量／金额，以及库存合计 |
| Inventory.checkRecord | 语法 | public InventoryCheckVO checkRecord( ) |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 显示截至盘点时间的库存快照 |
| Inventory.getGiftID | 语法 | public String getGiftID () |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 返回对应的库存赠送单的ID |
| Inventory.getOverFlowID | 语法 | public String getOverFlowID () |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 返回对应的库存报溢单的ID |
| Inventory.getLossID | 语法 | public String getLossID () |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 返回对应的库存报损单的ID |
| Inventory.getAlarmID | 语法 | public String getAlarmID () |
| 前置条件 | 已经得到库存单据数据 |
| 后置条件 | 返回对应的库存警报单的ID |
| Inventory.addCommodity | 语法 | public void addCommodity(String ID, int number) |
| 前置条件 | 完成库存的盘点，输入数据 |
| 后置条件 | 生成相应的库存报溢单 |
| Inventory.submit | 语法 | public InventoryBillVO submit(String remark) |
| 前置条件 | 输入数据合法 |
| 后置条件 | 生成对应的单据 |
| Inventory.save | 语法 | public InventoryBillVO save(String remark) |
| 前置条件 | 输入的数据合法 |
| 后置条件 | 保存对应的单据为草稿 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| InventoryDataService.  insert(InventoryBillPO po) | 生成单据，告知是否成功 | |
| InventoryDataService.  update(InventoryBillPO po) | 更改单据数据 | |

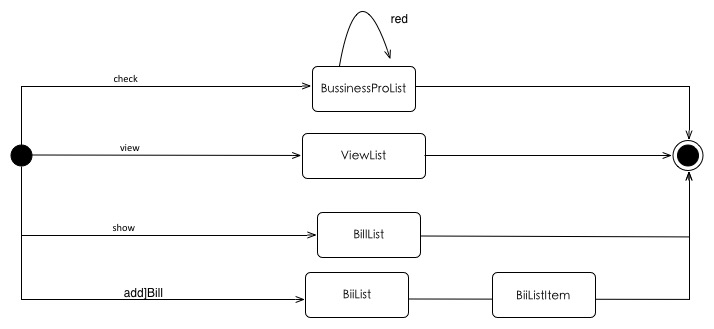
1. 业务逻辑层的动态模型

如图表明了当库存管理人员输入需要查看的时间段之后，库存业务逻辑处理的相关对象之间的协作

如图表明了当库存管理人员想要添加赠送单的时候的顺序图



如图所示的状态图描述了Inventory对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作



### Clientbl模块

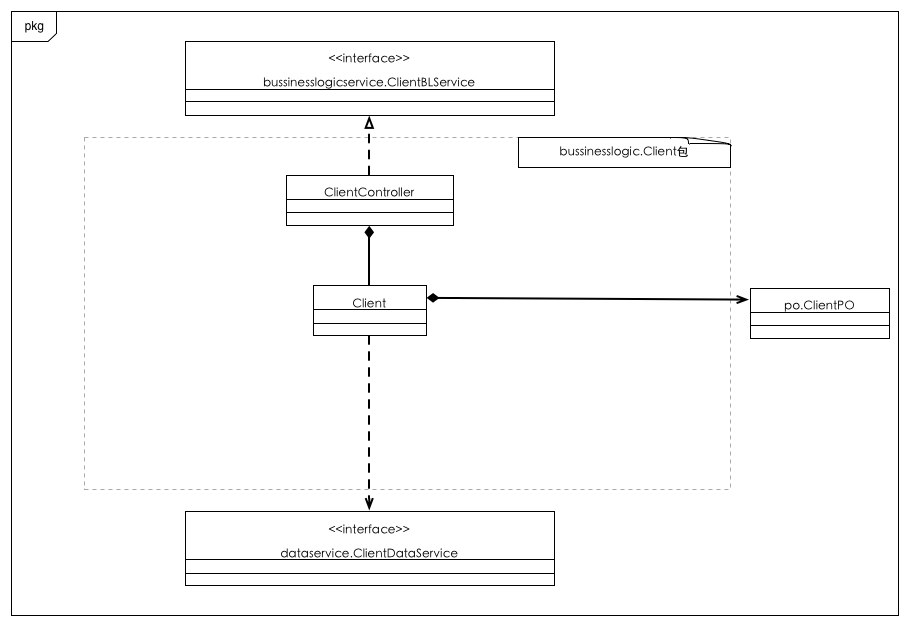
1. 模块概述

Clientbl模块负责客户的管理，创建新的客户，删除客户，修改客户，查找客户。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. ClientBLService接口。业务逻辑层和数据层之间添加dataservice. ClientDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ClientController，这样ClientController会将对客户的逻辑业务处理委托给Client对象。ClientPO是作为客户的持久化对象被添加到设计模型中去的。

Clientbl模块的设计如图



**clientbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| ClientController | 负责实现客户管理界面所需要的服务 |
| Client | 客户管理的领域模型对象，拥有客户数据的编号、分类、级别、姓名、电话、地址、邮编、电子邮箱、应收额度、应收、应付、默认业务员。 |

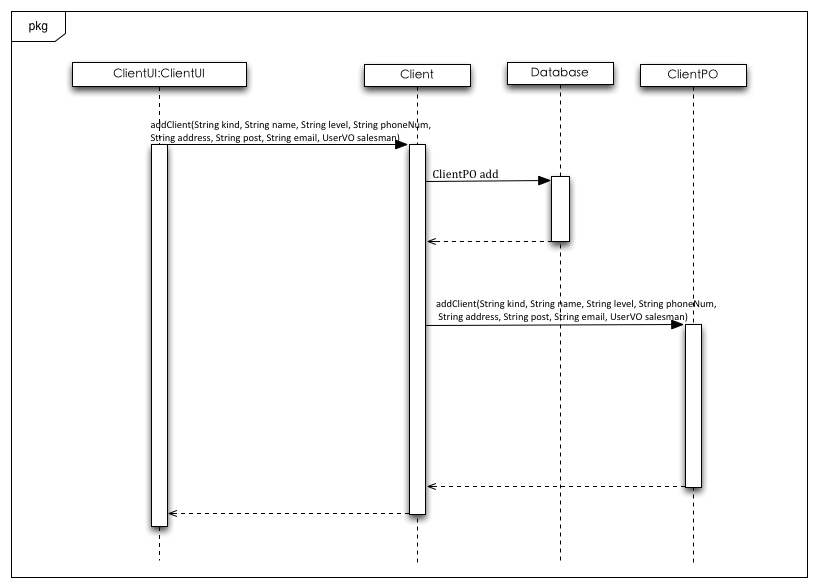
1. 模块内部类的接口规范

**Client的接口规范**

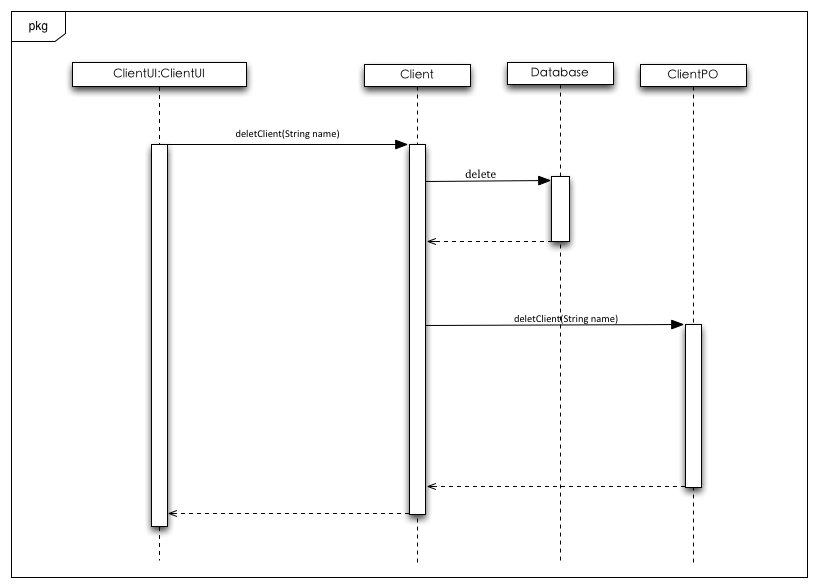
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Client.findClient | 语法 | public ResultMessage findClient(String keywords, long id) |
| 前置条件 | 关键字符合输入条件，id符合输入条件 |
| 后置条件 | 查找到目标客户 |
| Client.addClient | 语法 | public ResultMessage addClient(String kind, String name, String level, String phoneNum, String address, String post, String email, UserVO salesman) |
| 前置条件 | 类型、名字、级别、电话、地址、邮编符合输入条件 |
| 后置条件 | 添加客户，持久化更新涉及的对象的数据 |
| Client.updClient | 语法 | public ResultMessage updClient(String kind, String name, String level, String phoneNum, String address, String email, String post, UserVO salesman) |
| 前置条件 | 修改的属性符合输入条件 |
| 后置条件 | 客户属性被修改，持久化更新涉及的对象的数据 |
| Client.deletClient | 语法 | public ResultMessage deletClient(String name) |
| 前置条件 | 输入的客户名称符合输入条件 |
| 后置条件 | 删除该客户，持久化更新涉及的对象的数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| ClientDataService.  findClient(int id) | 根据ID进行查找单一持久化对象 | |
| ClientDataService.  findClients(String fields, int id) | 根据字段名和值进行查找多个持久化对象 | |
| ClientDataService.  add(ClientPO po) | 添加单一化持久对象 | |
| ClientDataService.  delet(ClientPO po) | 删除单一化持久对象 | |
| ClientDataService.  update(ClientPO po) | 更改单一化持久对象 | |
| ClientDataService.  getClientData | 得到Client数据的服务引用 | |

1. 业务逻辑层的动态模型

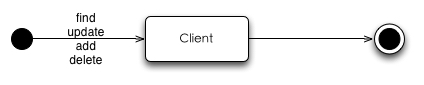
如图表明了进销存管理系统中，当进货销售人员想要添加一个客户的时候，客户管理业务逻辑处理的相关对象之间的协作。



如图表明了进销存管理系统中，当进货销售人员想要删除一个客户的时候，客户管理业务逻辑处理的相关对象之间的协作



如图所示的状态图描述了Client对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。



### Purchasebl模块

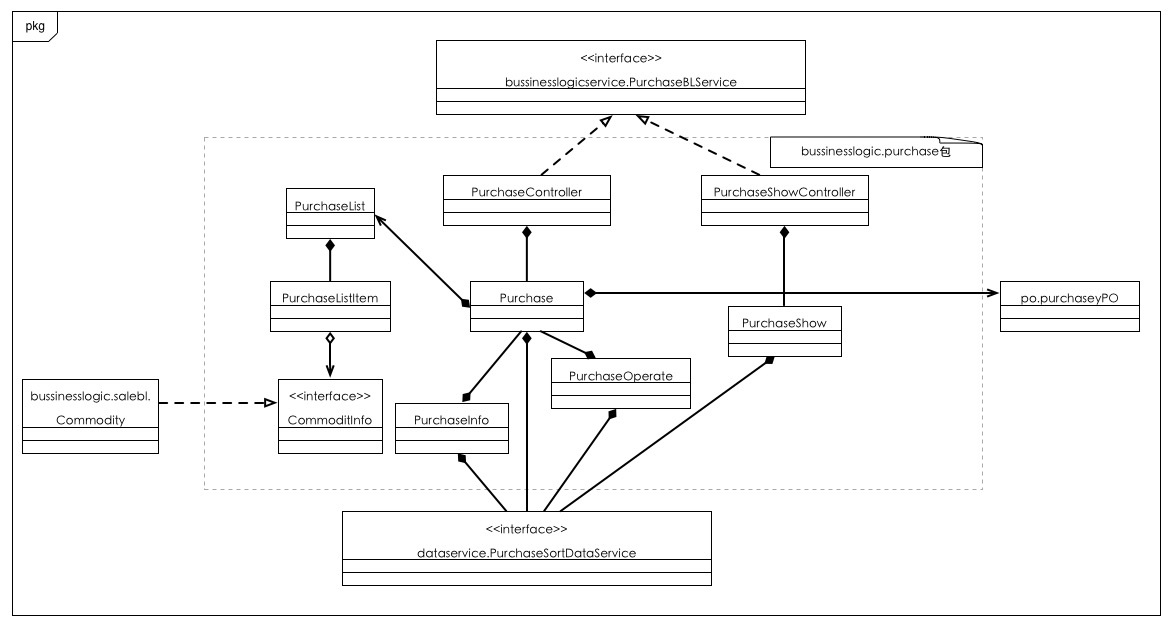
1. 模块概述

Purchasebl模块负责进货单、进货退货单的创建。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. PurchaseBLService接口。业务逻辑层和数据层之间添加dataservicePurchaseDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了PurchaseController，这样PurchaseController会将对进货的逻辑业务处理委托给Purchase对象。PurchasePO是作为进货单的持久化对象被添加到设计模型中去的。

如图展示了Purchaseb模块的设计



**purchasebl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| PurchaseController | 负责实现进货管理界面所需要的服务 |
| Purchase | 制定进货单的领域模型对象，拥有进货单的信息 |
| PurchaseShowController | 负责实现不同类型的进货单的展示 |
| PurchaseShow | 不同进货单展示的领域模型对象，拥有已存在的单据的信息 |
| PurchaseInfo | 负责为其他包提供进货单的数据 |
| PurchaseOperate | 负责进行其他包的操作造成的进货单数据的更改 |
| PurchaseList | 每一个进货单的数据 |
| PurchaseListItem | 进货单数据里面的条目 |
| PurchaseTrans | 转换进货的数据 |

1. 模块内部类的接口规范

**PurchaseController的接口规范**

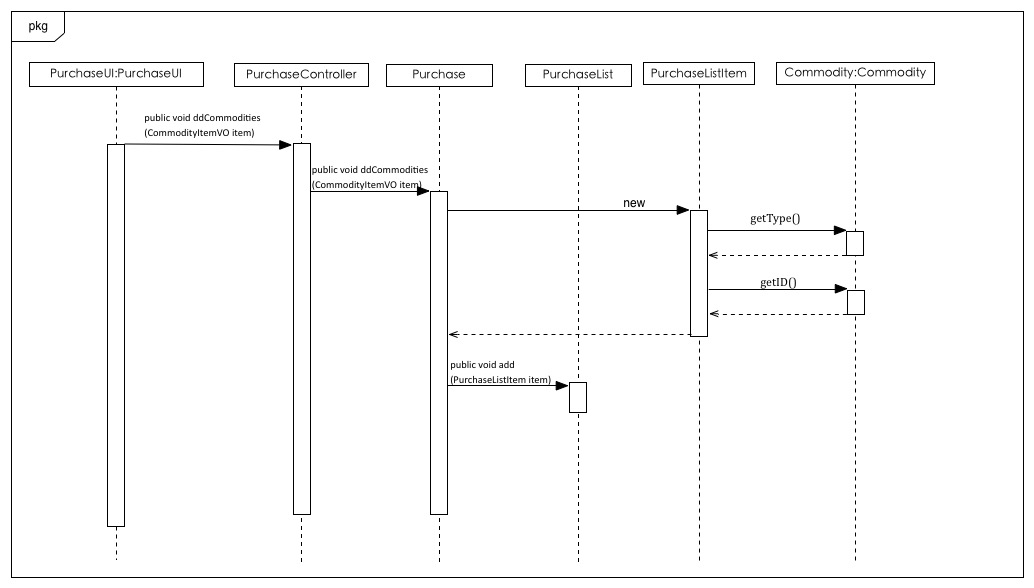
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PurchaseController.  getPurchaseID | 语法 | public String getPurchaseID () |
| 前置条件 | 创建一个Purchase对象 |
| 后置条件 | 调用Purchase的方法 |
| PurchaseController.  getPurbackID | 语法 | public String getPurbackID () |
| 前置条件 | 创建一个Purchase对象 |
| 后置条件 | 调用Purchase的方法 |
| PurchaseController.  addCommodities | 语法 | public void addCommodities  (CommodityItemVO item) |
| 前置条件 | 创建一个Purchase对象 |
| 后置条件 | 调用Purchase的方法 |
| PurchaseController.  submit | 语法 | public PurchaseVO submit(PurInputInfo info) |
| 前置条件 | 创建一个Purchase对象 |
| 后置条件 | 调用Purchase的方法 |
| PurchaseController.  save | 语法 | public PurchaseVO save(PurInputInfo info) |
| 前置条件 | 创建一个Purchase对象 |
| 后置条件 | 调用Purchase的方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Purchase.getPurchaseID() | 得到进货单的ID | |
| Purchase.getPurbackID() | 得到进货退货单的ID | |
| Purchase.addCommodities  (CommodityItemVO item) | 添加商品 | |
| Purchase.submit(PurInputInfo info) | 提交单据 | |
| Purchase.save(PurInputInfo info) | 保存为草稿 | |

**Purchase的接口规范**

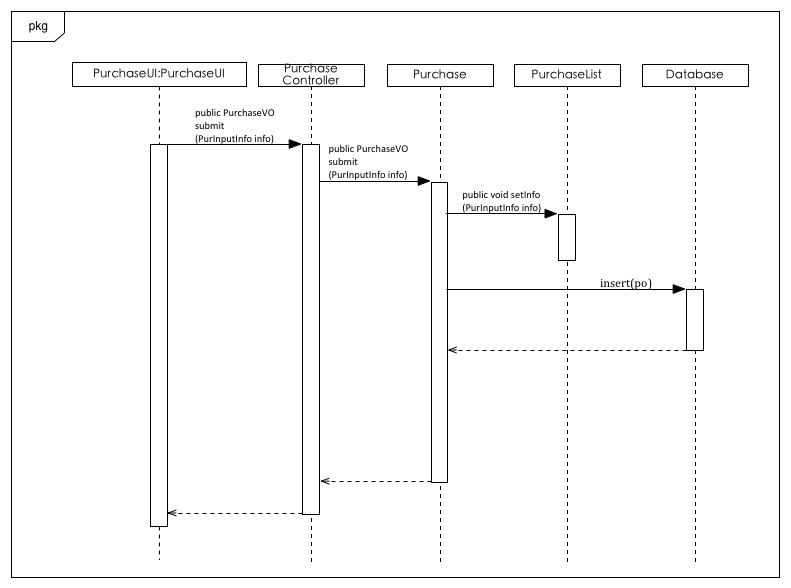
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Purchase.getPurchaseID | 语法 | public String getPurchaseID () |
| 前置条件 | 开始创建一个进货单 |
| 后置条件 | 得到该进货单的编号 |
| Purchase.getPurbackID | 语法 | public String getPurbackID () |
| 前置条件 | 开始创建一个进货退货单 |
| 后置条件 | 得到该进货退货单的编号 |
| Purchase.addCommodities | 语法 | public void addCommodities  (CommodityItemVO item) |
| 前置条件 | 商品名称、数量、单价、备注符合输入规则 |
| 后置条件 | 添加商品到商品清单 |
| Purchase.buildPur | 语法 | public PurchasePO buildPur() |
| 前置条件 | 添加相应的数据到进货单中 |
| 后置条件 | 生成进货单PO |
| Purchase.submit | 语法 | public PurchaseVO submit(PurInputInfo info) |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| Purchase.save | 语法 | public PurchaseVO save(PurInputInfo info) |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被保存为草稿 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| PurchaseDataService.  find(String id) | 根据ID进行查找单一持久化对象 | |
| PurchaseDataService.  insert(PurchasePO po) | 插入单一持久化对象 | |
| PurchaseDataService.  delete(PurchasePO po) | 删除单一持久化对象 | |
| PurchaseDataService.  update(PurchasePO po) | 更新单一持久化对象 | |
| PurchaseDataService.init | 清空单一持久化对象 | |

1. 业务逻辑层的动态模型

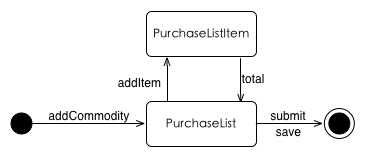
如图所示展示了当一名进货销售人员想要将商品添加进进货单时，需要进行的操作。



如图展示了进货人员需要进行单据的提交时



如图所示的状态图描述了Purchase对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。



### Salebl模块

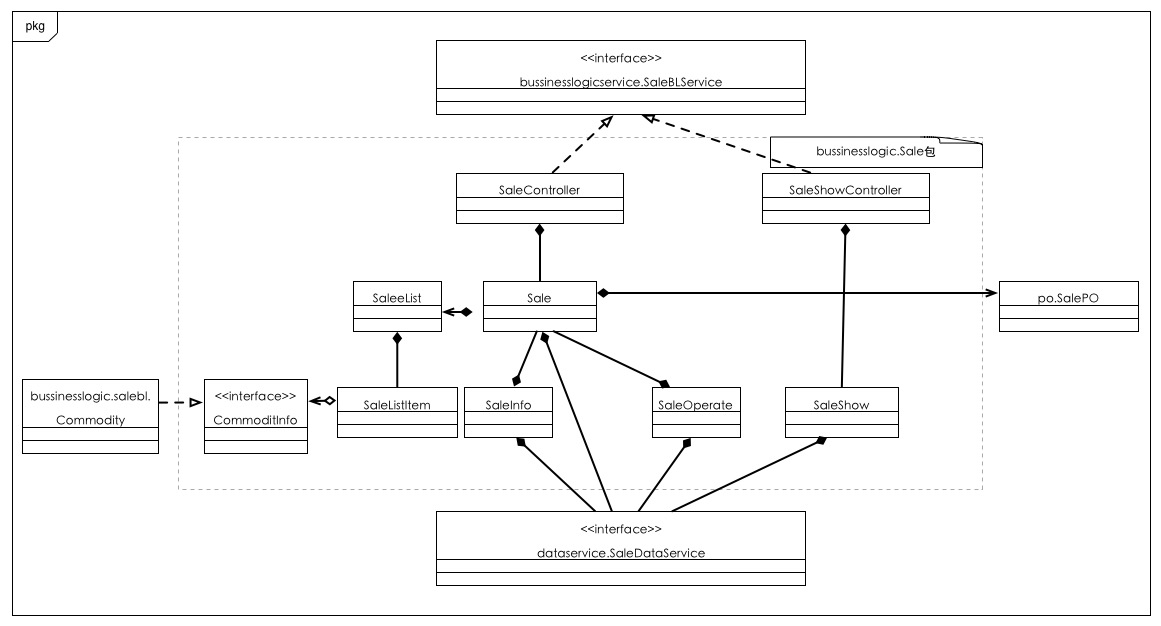
1. 模块概述

Salebl模块负责销售单、销售退货单的创建。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. SaleBLService接口。业务逻辑层和数据层之间添加dataservice. SaleDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了SaleController，这样SaleController会将对销售的逻辑业务处理委托给Sale对象。SalePO是作为销售单的持久化对象被添加到设计模型中去的。

Salebl模块的设计如图所示



**salebl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| SaleController | 负责实现销售管理界面所需要的服务 |
| Sale | 制定销售单的领域模型对象，拥有销售单的信息 |
| SaleShowController | 负责实现不同类型的销售单的展示 |
| SaleShow | 不同销售单展示的领域模型对象，拥有已存在的单据的信息 |
| SaleInfo | 负责为其他包提供销售单的数据 |
| SaleOperate | 负责进行其他包的操作造成的销售单数据的更改 |
| SaleList | 每一个销售单的数据 |
| SaleListItem | 销售单数据里面的条目 |
| SaleTrans | 转换销售的数据 |

1. 模块内部类的接口规范

**SaleController的接口规范**

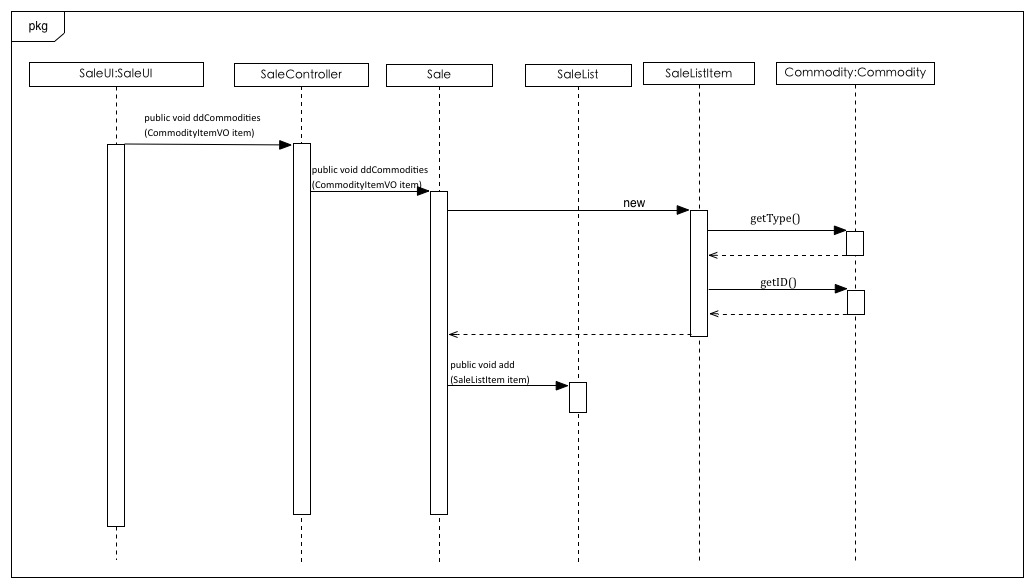
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| SaleController.  getSaleID | 语法 | public String getSaleID () |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.  getSalebackID | 语法 | public String getSalebackID () |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.  showBargains | 语法 | public ArrayList<PromotionBargainVO>  showBargains(); |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.  addCommodities | 语法 | public void addCommodities  (CommodityItemVO item) |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.addClient | 语法 | public void addClient(String clientID) |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.  findFitPromotionClient | 语法 | public ArrayList<PromotionClientVO>  findFitPromotionClient() |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.  findFitPromotionTotal | 语法 | public ArrayList<PromotionClientVO>  findFitPromotionTotal() |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.  submit | 语法 | public SaleVO submit(PurInputInfo info) |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.  save | 语法 | public SaleVO save(PurInputInfo info) |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Purchase.getPurchaseID() | 得到进货单的ID | |
| Purchase.getPurbackID() | 得到进货退货单的ID | |
| Purchase.addCommodities  (CommodityItemVO item) | 添加商品 | |
| Purchase.submit(PurInputInfo info) | 提交单据 | |
| Purchase.save(PurInputInfo info) | 保存为草稿 | |

**Sale的接口规范**

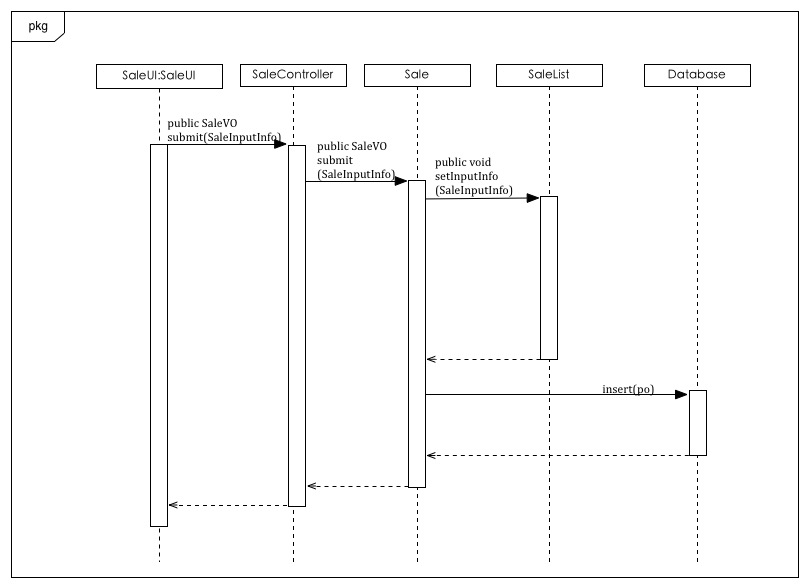
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Sale.getPurchaseID | 语法 | public String getPurchaseID () |
| 前置条件 | 开始创建一个销售单 |
| 后置条件 | 得到该销售单的编号 |
| Sale.getPurbackID | 语法 | public String getPurbackID () |
| 前置条件 | 开始创建一个销售退货单 |
| 后置条件 | 得到该销售退货单的编号 |
| Sale.showBargains | 语法 | public ArrayList<PromotionBargainVO>  showBargains(); |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| Sale.addCommodities | 语法 | public void addCommodities  (CommodityItemVO item) |
| 前置条件 | 商品名称、数量、单价、备注符合输入规则 |
| 后置条件 | 添加商品到商品清单 |
| SaleController.addClient | 语法 | public void addClient(String clientID) |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.  findFitPromotionClient | 语法 | public ArrayList<PromotionClientVO>  findFitPromotionClient() |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| SaleController.  findFitPromotionTotal | 语法 | public ArrayList<PromotionClientVO>  findFitPromotionTotal() |
| 前置条件 | 创建一个Sale对象 |
| 后置条件 | 调用Sale的方法 |
| Sale.buildPur | 语法 | public SaleVO buildPur() |
| 前置条件 | 添加相应的数据到销售单中 |
| 后置条件 | 生成进货单PO |
| Sale.submit | 语法 | public SaleVO submit(SaleInputInfo info) |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| Sale.save | 语法 | public SaleVO save(SaleInputInfo info) |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被保存为草稿 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| SaleDataService.  find(String id) | 根据ID进行查找单一持久化对象 | |
| SaleDataService.  insert(SalePO po) | 插入单一持久化对象 | |
| SaleDataService.  update(SalePO po) | 更新单一持久化对象 | |
| SaleDataService.init | 清空单一持久化对象 | |

1. 业务逻辑层的动态模型

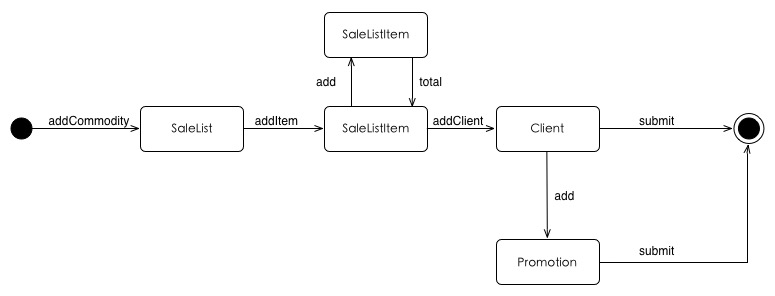
如图展示了进货销售人员想在销售单／销售退货单中添加一个商品的时候



如图展示了进货销售人员想要提交一个销售单／销售退货单



如图所示的状态图描述了Sale对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。



### Recordbl模块

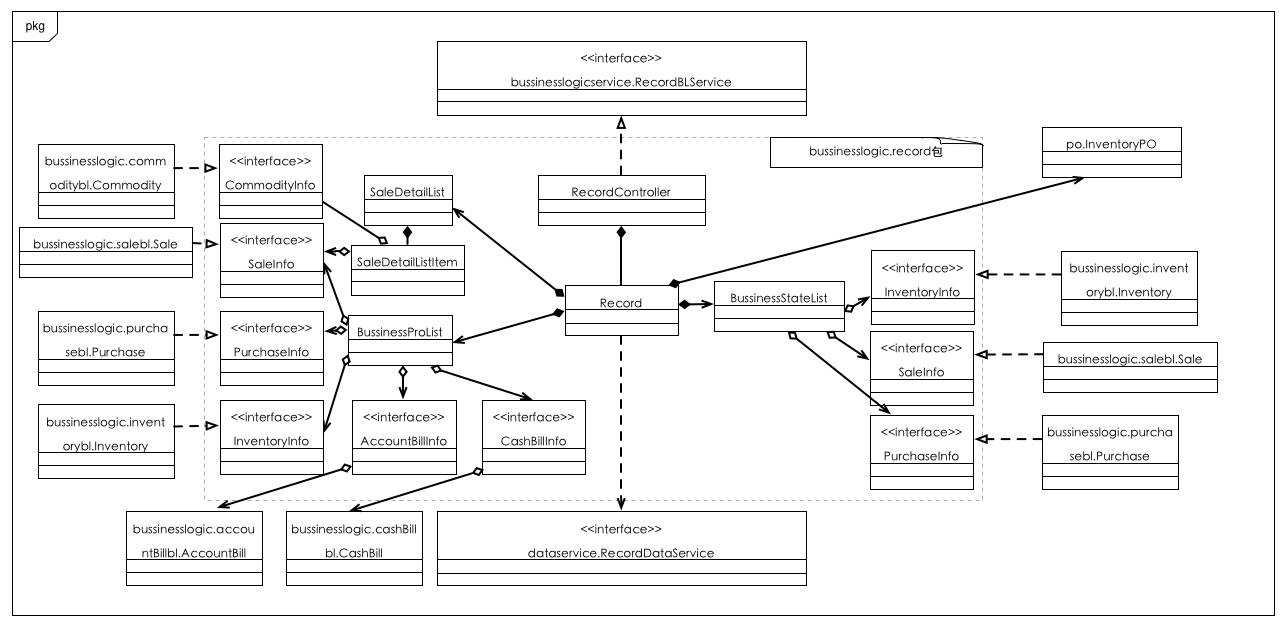
1. 模块概述

Recordbl模块负责进行销售明细表、经营情况表、经营历程表的查看。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. RecordBLService接口。业务逻辑层和数据层之间添加dataservice. RecordDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了RecordController，这样RecordController会将对报表的逻辑业务处理委托给Record对象。

Recordbl模块的设计如下



**recordbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| RecordController | 负责实现单据管理界面所需要的服务 |
| Record | 单据管理的领域模型对象，可以实现查看查看销售明细表统计一段时间内商品的销售情况，查看经营历程表，但是不可修改和删除。在此可以进行红冲和红冲并复制的操作。  查看经营情况表（统计显示一段时间内的经营收支状况和利润。经营收入显示为折让后，并显示出折让了多少。显示信息： |

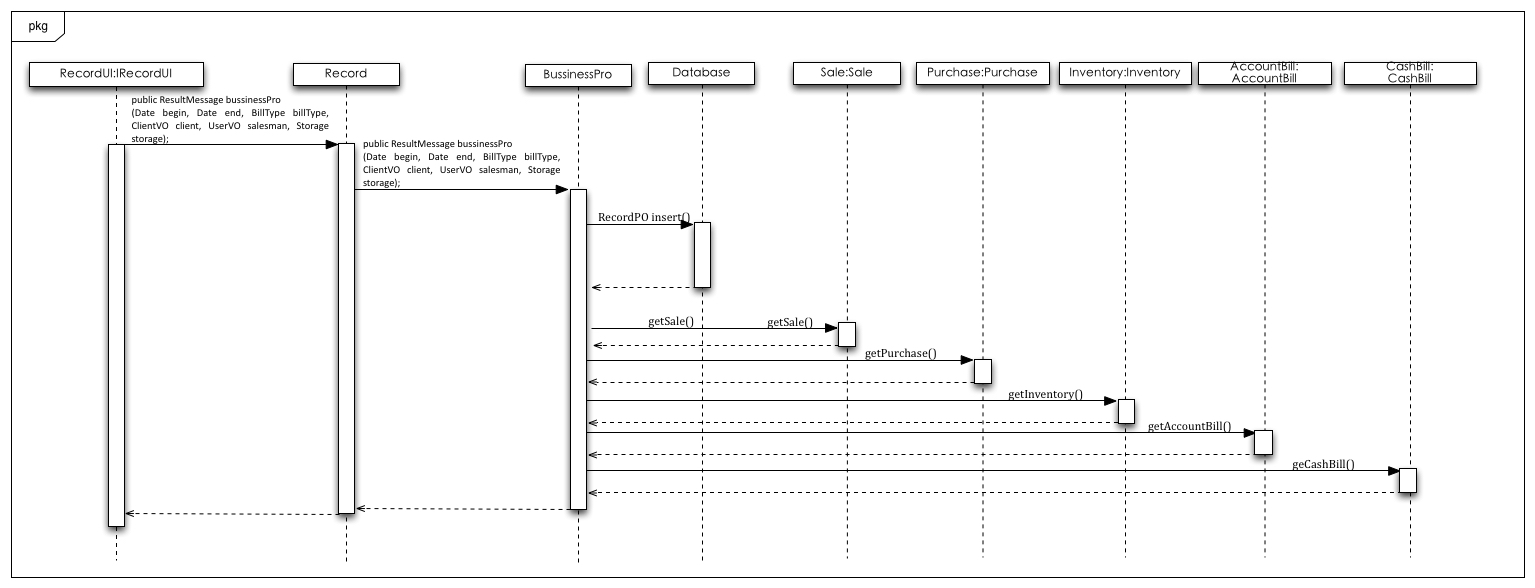
1. 模块内部类的接口规范

**Record的接口规范**

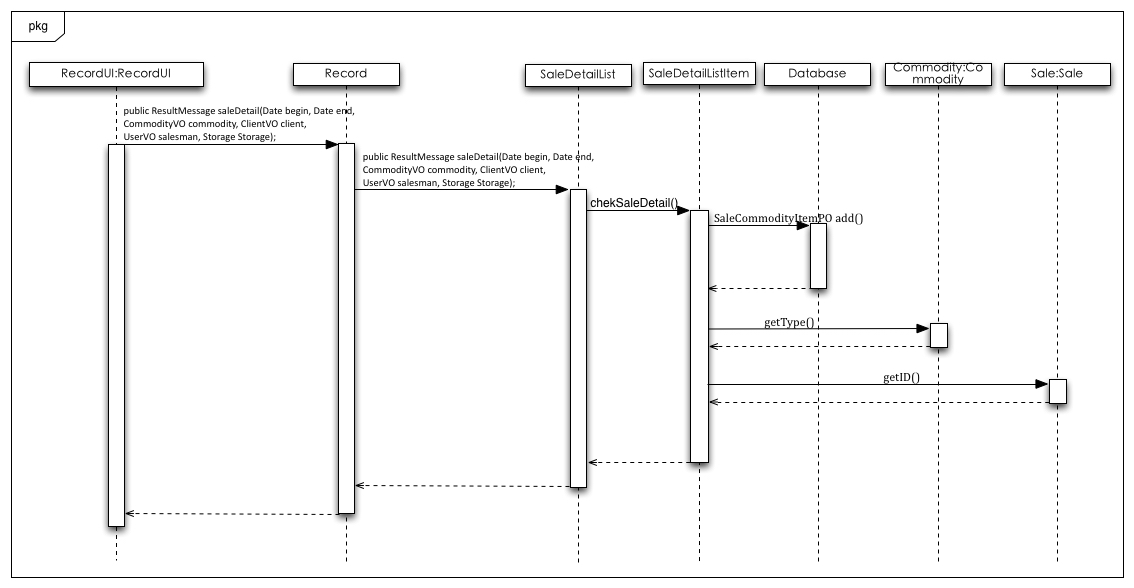
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Record.saleDetail | 语法 | public ResultMessage saleDetail(Date begin, Date end, CommodityVO commodity, ClientVO client, UserVO salesman, Storage Storage); |
| 前置条件 | 时间区间、商品名、客户、业务员、仓库符合输入规则 |
| 后置条件 | 返回符合条件的销售明细表 |
| Record.businessPro | 语法 | public ResultMessage bussinessPro  (Date begin, Date end, BillType billType, ClientVO client, UserVO salesman, Storage storage); |
| 前置条件 | 时间区间、单据类型、商品、业务员、仓库符合输入规则 |
| 后置条件 | 返回符合条件的单据的历程表 |
| Record.red | 语法 | public ResultMessage red  (ValueObject valueRecord) |
| 前置条件 | 有之前的单子做模板 |
| 后置条件 | 添加红冲 |
| Record.copyRed | 语法 | public ResultMessage copyRed  (ValueObject valueRecord) |
| 前置条件 | 有之前的单子做模板 |
| 后置条件 | 修改红冲 |
| Record.bussinessState | 语法 | public ResultMessage businessState  (Date begin, Date end) |
| 前置条件 | 时间区间符合输入规则 |
| 后置条件 | 返回符合条件的经营情况表 |
| Record.earning | 语法 | public ResultMessage earning() |
| 前置条件 | 完成单据的类型选择 |
| 后置条件 | 返回计算得到的收入 |
| Record.expense | 语法 | public ResultMessage expense() |
| 前置条件 | 完成单据的类型选择 |
| 后置条件 | 返回计算得到的支出 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| RecordDataService.  insert(RecordPO po) | 插入单一持久化对象 | |
| RecordDataService.  delete(RecordPO po) | 删除单一持久化对象 | |
| RecordDataService.  update(RecordPO po) | 更新单一持久化对象 | |

1. 业务逻辑层的动态模型

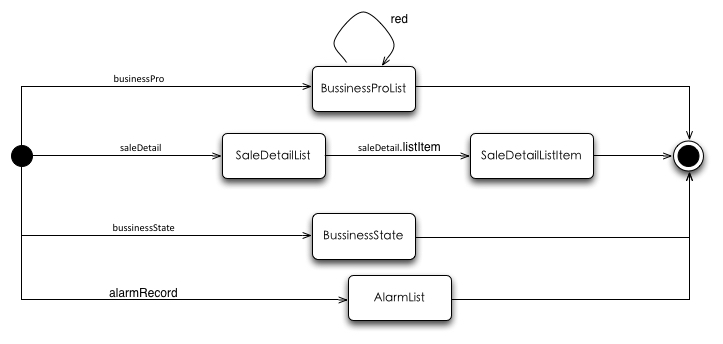
如图展示了进货销售人员想要查看经营历程表时的操作。



如图展示了进货销售人员想要查看销售明细表时的操作



如图所示的状态图描述了Record对象的生存期间的状态序列、引起转移的事件，以及因状态转移而伴随的动作。



### Accountbl模块

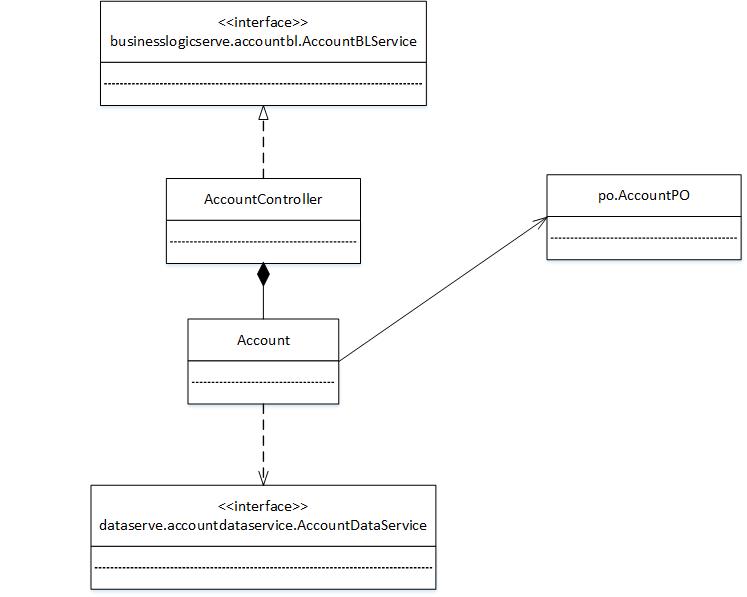
1. 模块概述

Accountbl模块负责进行公司账户的管理，对账户进行创建、更改、删除、查找。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. AccountBLService接口。业务逻辑层和数据层之间添加dataservice. AccountDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了AccountController，这样AccountController会将对账户的逻辑业务处理委托给Account对象。AccountPO是作为账户的持久化对象被添加到设计模型中去的。

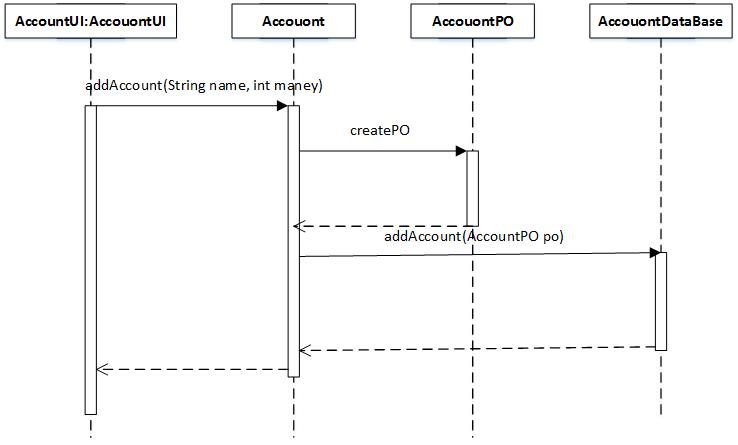
accountbl模块的设计如图



1. 模块内部类的接口规范

**Account的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Account.findAccount | 语法 | public ResultMessage findAccount  (String keyword) |
| 前置条件 | keyword符合输入规则 |
| 后置条件 | 返回查找的账户 |
| Account.addAccount | 语法 | public ResultMessage addAccount(String name, int money) |
| 前置条件 | name和money符合输入规则 |
| 后置条件 | 增加新的账户，持久化更新涉及的对象的数据 |
| Account.deletAccount | 语法 | public ResultMessage deleteAccount  (String name) |
| 前置条件 | name符合输入规则 |
| 后置条件 | 删除账户，持久化更新涉及的对象的数据 |
| Account.updAccount | 语法 | public ResultMessage updAccount(String name) |
| 前置条件 | name符合输入规则 |
| 后置条件 | 更改账户，持久化更新涉及的对象的数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| AccountDataService.  find(String id) | 根据ID进行查找单一持久化对象 | |
| AccountDataService.  insert(AccountPO po) | 插入单一持久化对象 | |
| AccountDataService.  delete(AccountPO po) | 删除单一持久化对象 | |
| AccountDataService.  update(AccountPO po) | 更新单一持久化对象 | |
| AccountDataService.init() | 清空单一持久化对象 | |
| AccountDataService.getClientData | 得到Account数据的服务引用 | |

1. 业务逻辑层的动态模型

下图是添加账户的顺序图和状态图



### AccountBillbl模块

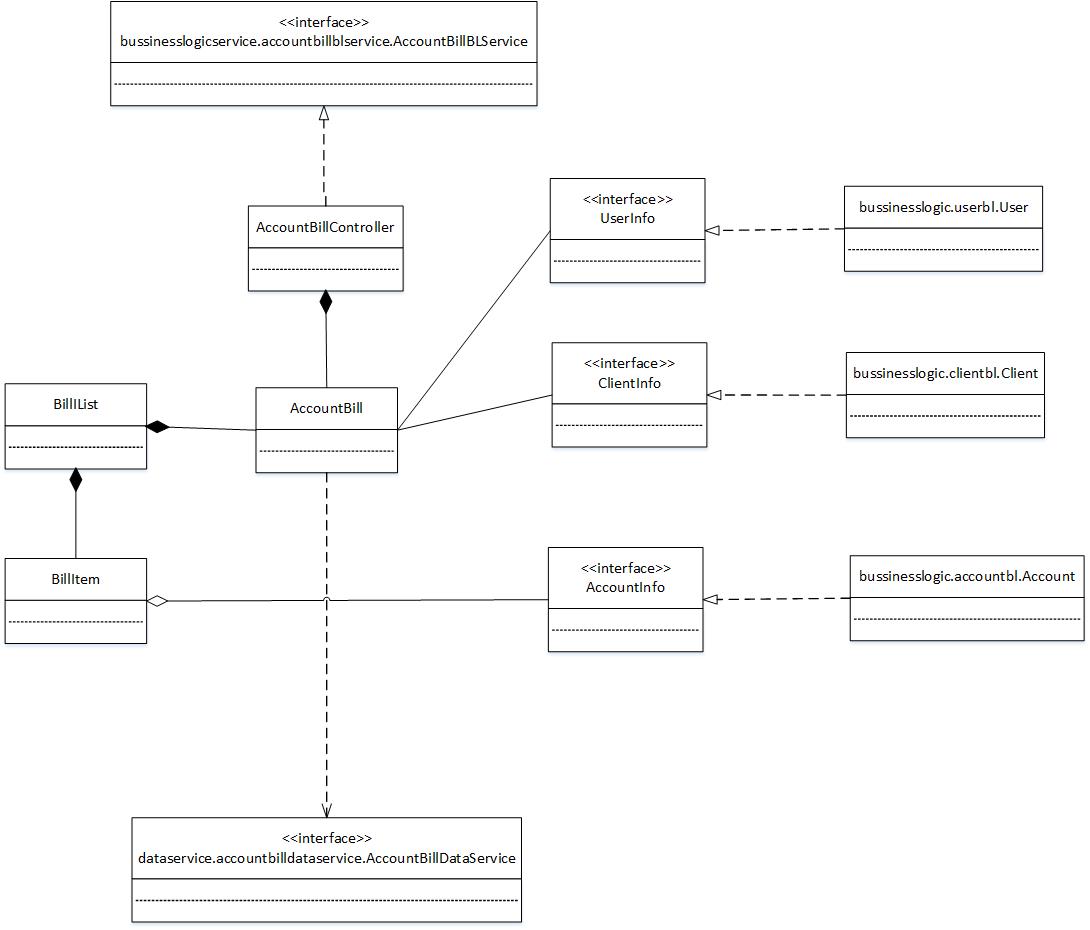
1. 模块概述

AccountBillbl模块负责建立收款单、付款单。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. AccountBillBLService接口。业务逻辑层和数据层之间添加dataservice. AccountBillDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了AccountBillController，这样AccountBillController会将对收付款的逻辑业务处理委托给AccountBill对象。AccountBillPO是作为收付款单的持久化对象被添加到设计模型中去的。

AccountBillbl模块类的设计



**AccountBillbl模块各个类的职责**

|  |  |
| --- | --- |
| 模块 | 职责 |
| AccountBillController | 负责实现收款单、付款单管理界面所需要的服务 |
| AccountBill | 收款单、付款单领域模型对象，可以实现建立收款单、付款单 |
| AccountBillShowController | 负责实现收款单、付款单的显示所需要的服务 |
| AccountBillShow | 收款单、付款单的显示的领域模型对象 |
| AccountBillInfo | 负责实现其他包对收、付款数据的显示、更改 |
| AccountBillList | 收款、付款单的数据条目 |
| AccountBillTrans | 负责实现收款、付款单的数据转换 |

1. 模块内部类的接口规范

**AccountBillController的接口规范**

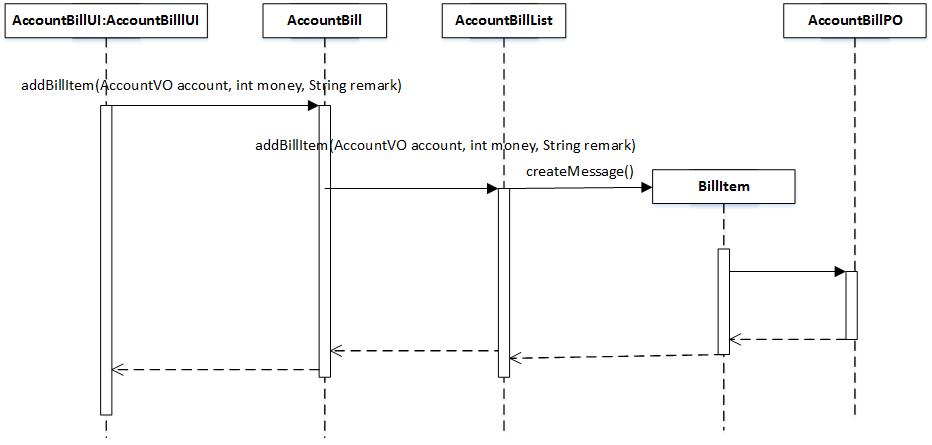
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountBillController  .getPayID | 语法 | public String getPayID() |
| 前置条件 | 得到数据，启动新建一个付款单 |
| 后置条件 | 返回新建的付款单的ID |
| AccountBillController  .getExpenseID | 语法 | public String getExpenseID() |
| 前置条件 | 得到数据，启动新建一个首款DNA |
| 后置条件 | 返回新建的收款单的ID |
| AccountBillController  .find | 语法 | public AccountBillVO find(String ID) |
| 前置条件 | 得到数据，输入合法 |
| 后置条件 | 返回根据ID找到的单据 |
| AccountBillController  .getAllClients | 语法 | public HashMap<String, String> getAllClients() |
| 前置条件 | 得到数据，需要显示所有客户 |
| 后置条件 | 返回所有的客户 |
| AccountBillController  .getAllAccounts | 语法 | public HashMap<String, String> getAllAccounts() |
| 前置条件 | 得到数据，需要显示所有的账户 |
| 后置条件 | 返回所有的账户 |
| AccountBillController  .addBillItem | 语法 | public ResultMessage addBillItem  (AccountBillItemVO vo) |
| 前置条件 | 符合输入规则 |
| 后置条件 | 添加转账列表 |
| AccountBillController  .getSumMoney | 语法 | public ResultMessage getSumMoney() |
| 前置条件 | 已经建立转账列表 |
| 后置条件 | 得到总金额 |
| AccountBillController  .addAccountBill | 语法 | public ResultMessage addAccountBill  (String clientID, String clientName) |
| 前置条件 | 客户编号、客户名称、账户、符合输入规则 |
| 后置条件 | 建立收款单 |
| AccountBillController  .submit | 语法 | public AccountBillVO submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| AccountBillController  .save | 语法 | public AccountBillVO save() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据保存为草稿 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| AccountBill.getPayID | 返回新建的付款单的ID | |
| AccountBill.getExpenseID | 返回新建的收款单的ID | |
| AccountBill.find | 返回根据ID查找得到的单据 | |
| AccountBill.getAllClients | 返回所有的客户 | |
| AccountBill.addBillItem | 添加账户条目 | |
| AccountBill.getSumMoney | 得到总价 | |
| AccountBill.addAccountBill | 添加一个单据 | |
| AccountBill.submit | 提交单据 | |
| AccountBill.save | 保存为草稿 | |

**AccountBill的接口规范**

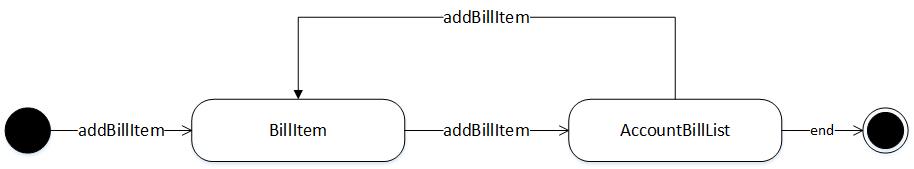
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountBill.getPayID | 语法 | public String getPayID() |
| 前置条件 | 得到数据，启动新建一个付款单 |
| 后置条件 | 返回新建的付款单的ID |
| AccountBill.getExpenseID | 语法 | public String getExpenseID() |
| 前置条件 | 得到数据，启动新建一个首款DNA |
| 后置条件 | 返回新建的收款单的ID |
| AccountBill.find | 语法 | public AccountBillVO find(String ID) |
| 前置条件 | 得到数据，输入合法 |
| 后置条件 | 返回根据ID找到的单据 |
| AccountBill.getAllClients | 语法 | public HashMap<String, String> getAllClients() |
| 前置条件 | 得到数据，需要显示所有客户 |
| 后置条件 | 返回所有的客户 |
| AccountBill.getAllAccounts | 语法 | public HashMap<String, String> getAllAccounts() |
| 前置条件 | 得到数据，需要显示所有的账户 |
| 后置条件 | 返回所有的账户 |
| AccountBill.addBillItem | 语法 | public ResultMessage addBillItem  (AccountBillItemVO vo) |
| 前置条件 | 符合输入规则 |
| 后置条件 | 添加转账列表 |
| AccountBill.getSumMoney | 语法 | public ResultMessage getSumMoney() |
| 前置条件 | 已经建立转账列表 |
| 后置条件 | 得到总金额 |
| AccountBill.addAccountBill | 语法 | public ResultMessage addAccountBill  (String clientID, String clientName) |
| 前置条件 | 客户编号、客户名称、账户、符合输入规则 |
| 后置条件 | 建立收款单 |
| AccountBill.submit | 语法 | public AccountBillVO submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| AccountBill.save | 语法 | public AccountBillVO save() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据保存为草稿 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| AccountBillDataService.  find(String id) | 根据查找单一持久化对象 | |
| AccountBillDataService.  insert(AccountBillPO po) | 插入单一持久化对象 | |
| AccountBillDataService.  delete(AccountBillPO po) | 删除单一持久化对象 | |
| AccountBillDataService.  update(AccountBillPO po) | 更新单一持久化对象 | |
| AccountBillDataService.init | 初始化单一持久化对象 | |

1. 业务逻辑层的动态模型

如下为添加收款单和付款单的顺序图和状态图



顺序图



状态图

### CashBillbl模块

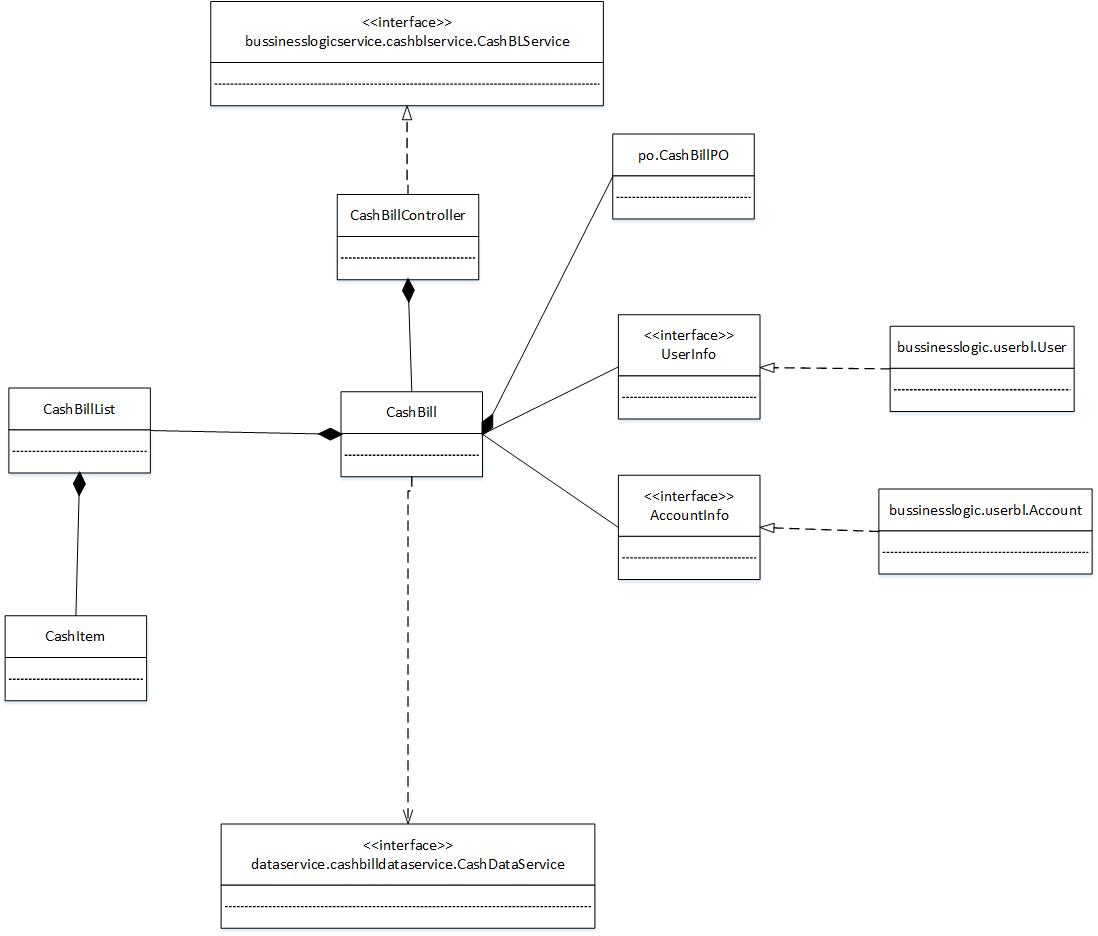
1. 模块概述

CashBillbl模块负责建立现金费用单具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. CashBillBLService接口。业务逻辑层和数据层之间添加dataservice. CashBillDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了CashBillController，这样CashBillController会将对现金单的逻辑业务处理委托给CashBill对象。CashBill是作为现金单的持久化对象被添加到设计模型中去的。

CashBillbl模块的设计如下图



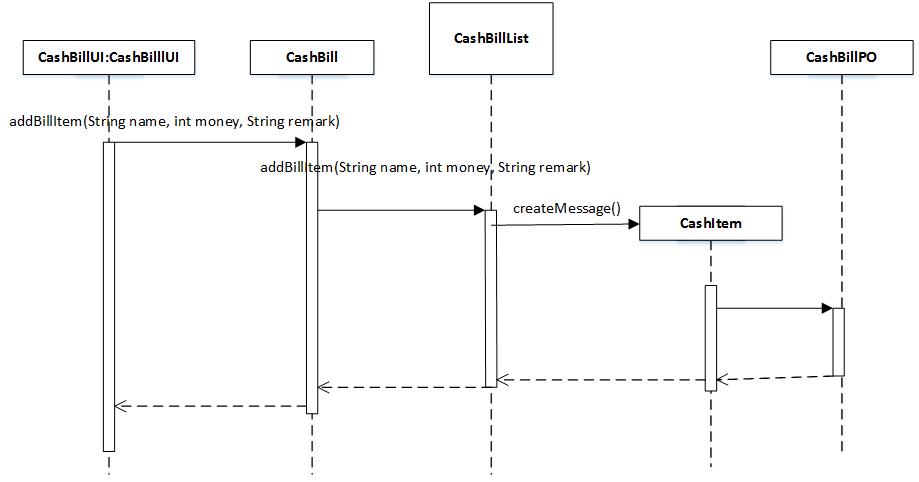
1. 模块内部类的接口规范

**CashBill的接口规范**

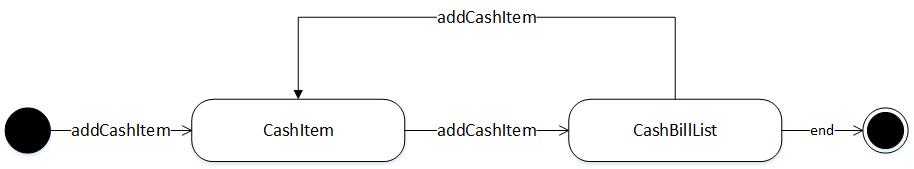
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| cashBill.addBillItem | 语法 | public ResultMessage addBillItem(String name, int money, String remark) |
| 前置条件 | 账户名、转账金额、备注符合输入规则 |
| 后置条件 | 建立转账列表 |
| cashBill.getSumMoney | 语法 | public ResultMessage getSumMoney  (ArrayList< CashItemVO > bills); |
| 前置条件 | 已经建立转账列表 |
| 后置条件 | 得到总金额 |
| cashBill. addCashBill | 语法 | public ResultMessage addCashBill (String id, UserVO user, AccountVO account, ArrayList<CashItemVO> bills, int sumMoney); |
| 前置条件 | 编号、客户、账户、总金额、备注符合输入规则 |
| 后置条件 | 建立收款单 |
| cashBill.submit | 语法 | public ResultMessage submit() |
| 前置条件 | 单据已经被制定出 |
| 后置条件 | 单据被提交发送给经理进行审批 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| CashBillDataService.  find(String id) | 根据查找单一持久化对象 | |
| CashBillDataService.  insert(CashBillPO po) | 插入单一持久化对象 | |
| CashBillDataService.  delete(CashBillPO po) | 删除单一持久化对象 | |
| CashBillDataService.  update(CashBillPO po) | 更新单一持久化对象 | |
| CashBillDataService.init | 初始化单一持久化对象 | |

1. 业务逻辑层的动态模型

下图为创建现金费用单的顺序图和状态图



创建现金费用单的顺序图



创建现金费用单的状态图

### Accountainitbl模块

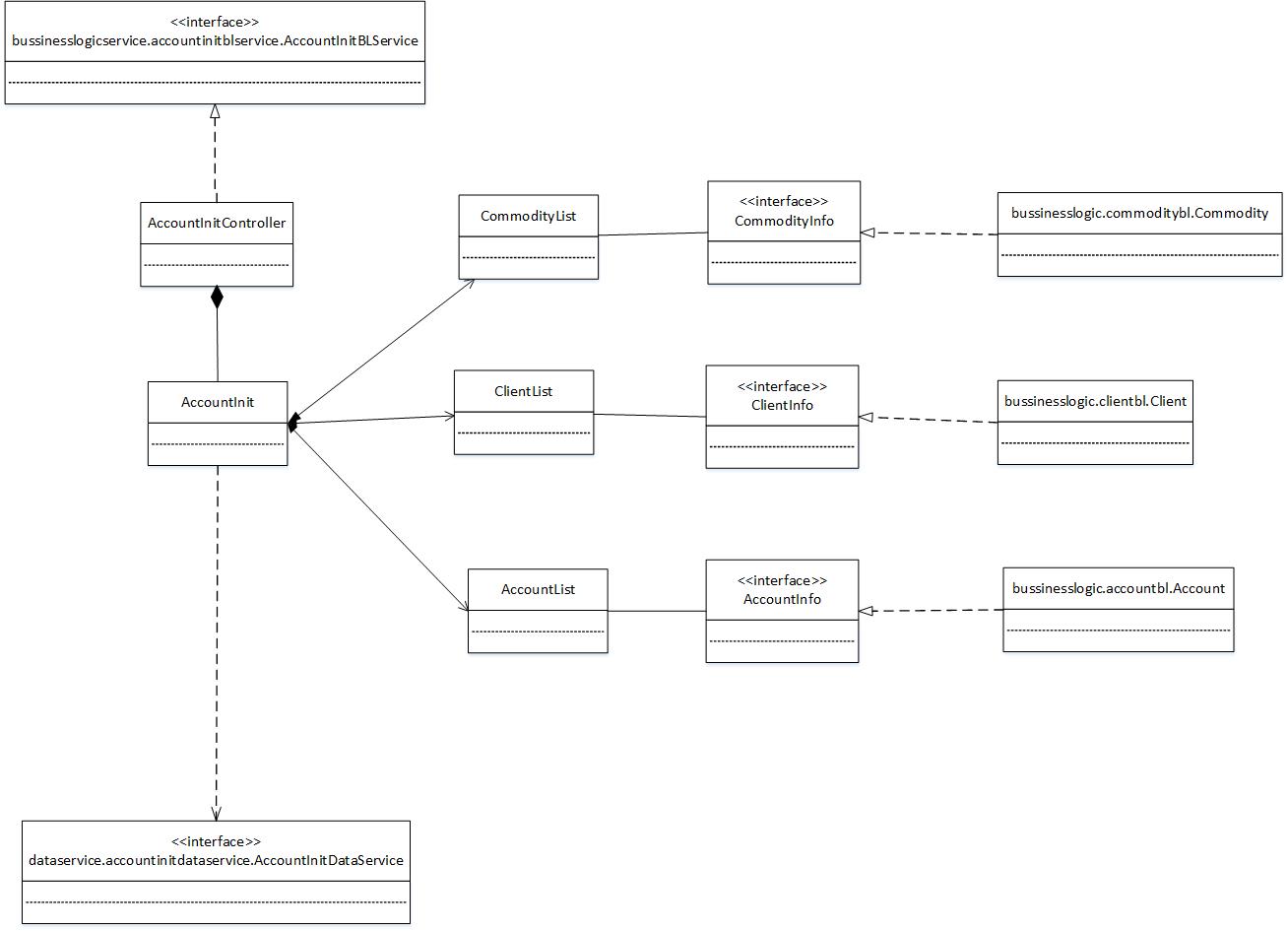
1. 模块概述

Accountainitbl模块负责实现建账界面所需要的服务,具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. AccountainitBLService接口。业务逻辑层和数据层之间添加dataservice. AccountainitDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了AccountainitController，这样AccountainitController会将对期初建账的逻辑业务处理委托给Accountainit对象。AccountainitPO是作为期初建账的持久化对象被添加到设计模型中去的。

AccountInitbi模块类的设计



Accountainitbl模块各个类的职责如下表

|  |  |
| --- | --- |
| 模块 | 职责 |
| AccountainitController | 负责实现期初建账界面所需要的服务 |
| Accountainit | 期初建账的领域模型对象，拥有期初建账的信息，可以帮助完成期初建账界面所需要的服务 |

1. 模块内部类的接口规范

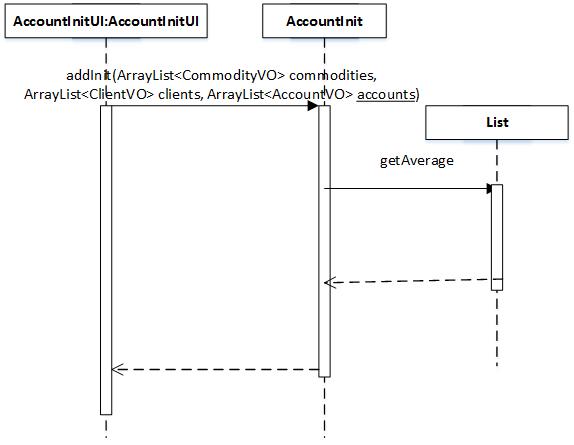
**AccountainitController接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountainitController.addInit | 语法 | public ResultMessage addInit  (ArrayList<CommodityVO> commodities,  ArrayList<ClientVO> clients,  ArrayList<AccountVO> accounts) |
| 前置条件 | 已经创建一个Accountainit领域对象,商品、客户、银行账户符合输入规范 |
| 后置条件 | 调用Accountainit的addInit方法 |
| AccountainitController.getEverage | 语法 | public ResultMessage getEverage  (int time, double purPrice, double salePrice) |
| 前置条件 | 已经创建一个Accountainit领域对象,已经添加了商品信息 |
| 后置条件 | 调用Accountainit的getEverage方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Accountainit.addInit (AccountaInitPO po) | 插入单一持久化对象 | |
| Accountainit. getEverage | 返回上年的平均进价和售价 | |

**Accountainit接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountaInit.addInit | 语法 | public ResultMessage addInit  (ArrayList<CommodityVO> commodities,  ArrayList<ClientVO> clients,  ArrayList<AccountVO> accounts) |
| 前置条件 | 商品、客户、银行账户符合输入规范 |
| 后置条件 | 建立期初账目 |
| AccountaInit.getEverage | 语法 | public ResultMessage getEverage  (int time, double purPrice, double salePrice) |
| 前置条件 | 已经添加了商品信息 |
| 后置条件 | 返回上年的平均进价和售价 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| AccountaInitDataService.  insert(AccountaInitPO po) | 插入单一持久化对象 | |
| AccountaInitDataService.  init | 初始化持久数据 | |

1. 下图是添加一个期初建账的顺序图和状态图



### Approvalbl模块

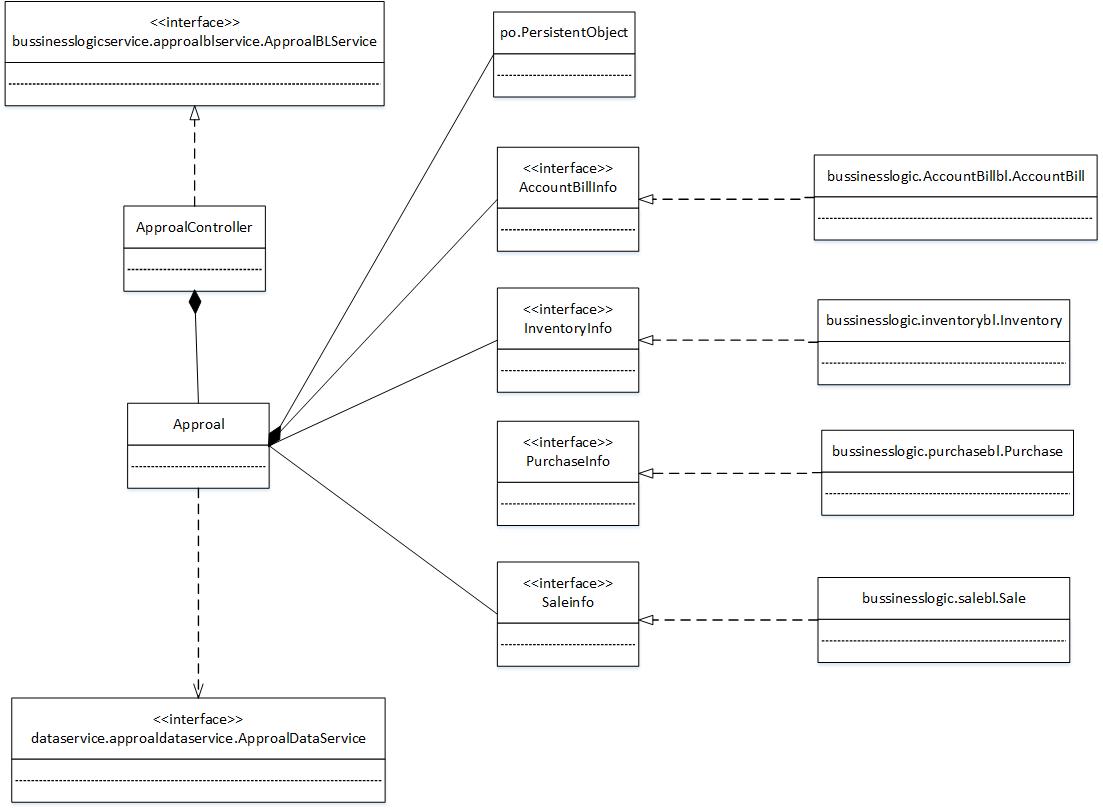
1. 模块概述

Approalbl模块负责实现审批界面所需要的服务。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. ApproalBLService接口。业务逻辑层和数据层之间添加dataservice. ApproalDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了ApproalController，这样ApproalController会将对促销的逻辑业务处理委托给Approal对象。

Approalbl模块的设计如图



Approalbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| ApproalController | 负责实现审批管理界面所需要的服务 |
| Approal | 审批的领域模型对象，拥有审批单子的信息，可以帮助完成审批管理界面所需要的服务 |

1. 模块内部类的接口规范

**ApproalController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ApproalController.checkRecord | 语法 | public ResultMessage checkRecord  (PersistentObject record) |
| 前置条件 | 已经创建一个Approal领域对象,有需要审核的单据 |
| 后置条件 | 调用Approal领域对象的checkRecord方法 |
| ApproalController.updRecord | 语法 | public ResultMessage updRecord  (PersistentObject record) |
| 前置条件 | 已经创建一个Approal领域对象,已经查看到单据内容 |
| 后置条件 | 调用Approal领域对象的updRecord方法 |
| ApproalController.submitRecord | 语法 | public ResultMessage submitRecord  (PersistentObject record) |
| 前置条件 | 已经创建一个Approal领域对象,确认单据内容没有错误 |
| 后置条件 | 调用Approal领域对象的submitRecord方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| Approal. checkRecord | 查看单据内容 | |
| Approal. updRecord | 修改单据内容 | |
| Approal. submitRecord | 确认单据内容没有错误 | |

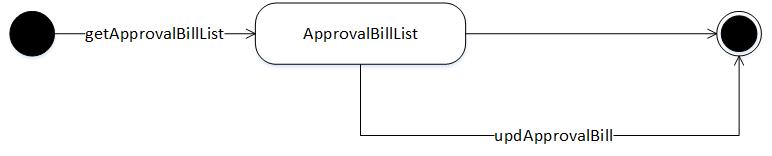
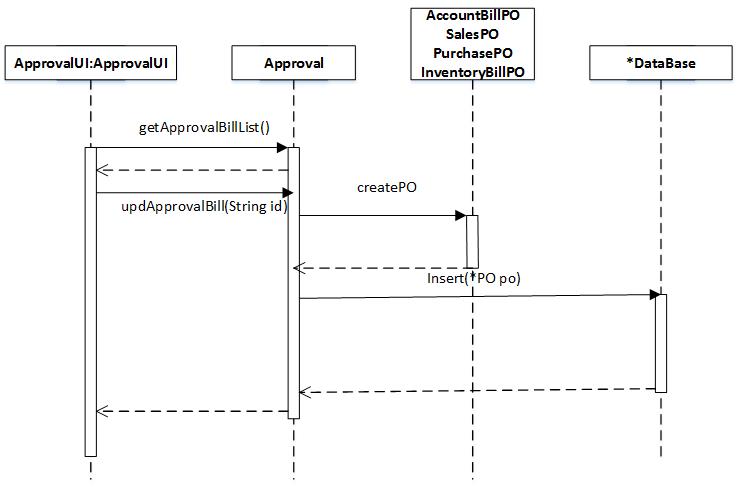
**Approal的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| Apporval.checkRecord | 语法 | public ResultMessage checkRecord  (PersistentObject record) |
| 前置条件 | 有需要审核的单据 |
| 后置条件 | 查看单据内容 |
| Apporval.updRecord | 语法 | public ResultMessage updRecord  (PersistentObject record) |
| 前置条件 | 已经查看到单据内容 |
| 后置条件 | 修改单据内容 |
| Apporval.submitRecord | 语法 | public ResultMessage submitRecord  (PersistentObject record) |
| 前置条件 | 确认单据内容没有错误 |
| 后置条件 | 通过单据，持久化更新涉及的对象的数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| ApprovalDataService.  Insert(ApprovalPO po) | 插入单一持久化对象 | |

1. 业务逻辑层的动态模型

下图为审批的顺序图和状态图。

getApprovalBill被ui调用，然后进入界面进入显示ApprovalBillList状态，可以审批并更新单据或者直接结束审批。



### Promotionbl模块

1. 模块概述

Promotionbl模块负责实现对促销策略的管理。具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice. PromotionBLService接口。业务逻辑层和数据层之间添加dataservice. PromotionDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了PromotionController，这样PromotionController会将对促销的逻辑业务处理委托给Promotion对象。PromotionPO是作为促销策略的持久化对象被添加到设计模型中去的。

Promotionbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| PromotionBargainController | 负责实现特价包促销策略管理界面所需要的服务 |
| PromotionBargain | 特价包促销策略的领域模型对象，拥有特价包促销策略的信息，可以帮助完成特价包促销策略管理界面所需要服务 |
| PromotionClientController | 负责实现客户等级促销策略管理界面所需要的服务 |
| PromotionClient | 客户等级促销策略的领域模型对象，拥有客户等级促销策略的信息，可以帮助完成客户等级促销策略管理界面所需要服务 |
| PromotionTotalController | 负责实现总价促销策略管理界面所需要的服务 |
| PromotionTotal | 总价促销策略的领域模型对象，拥有总价促销策略的信息，可以帮助完成总价促销策略管理界面所需要服务 |
| Promotion | 促销策略的父类，负责得到数据层的数据 |
| PromotionInfo | 负责为其他包调用促销策略的内容 |
| PromotionList | 促销策略的内容 |
| PromotionListItem | 促销策略的条目 |
| PromotionTrans | 促销策略的数据转换 |

1. 模块内部类的接口规范

**PromotionBargainController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionBargianController  .show | 语法 | public ArrayList<PromotionBargainVO> show() |
| 前置条件 | 创建了一个PromotionBargain对象 |
| 后置条件 | 调用PromotionBargain的方法 |
| PromotionBargianController. addBargain | 语法 | public void addBargain(CommodityItemVO vo) |
| 前置条件 | 创建了一个PromotionBargain对象 |
| 后置条件 | 调用PromotionBargain的方法 |
| PromotionBargianController. setBargainTotalPrice | 语法 | public void setBargainTotalPrice(double price) |
| 前置条件 | 创建了一个PromotionBargain对象 |
| 后置条件 | 调用PromotionBargain的方法 |
| PromotionBargianController. submit | 语法 | public PromotionBargainVO submit  (String beginDate, String endDate) |
| 前置条件 | 创建了一个PromotionBargain对象 |
| 后置条件 | 调用PromotionBargain的方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| PromotionBargian.show | 展示所有特价包 | |
| PromotionBargian. addBargain | 添加商品到特价包中 | |
| PromotionBargian. setBargainTotalPrice | 设置特价包的总价 | |
| PromotionBargian. submit | 提交特价包 | |

**PromotionBargain的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionBargian.show | 语法 | public ArrayList<PromotionBargainVO> show() |
| 前置条件 | 得到数据 |
| 后置条件 | 返回所有的特价包 |
| PromotionBargian. addBargain | 语法 | public void addBargain(CommodityItemVO vo) |
| 前置条件 | 符合输入规范 |
| 后置条件 | 添加商品到特价包中 |
| PromotionBargian. setBargainTotalPrice | 语法 | public void setBargainTotalPrice(double price) |
| 前置条件 | 特价包中的商品添加完成 |
| 后置条件 | 设置特价包的总价 |
| PromotionBargian. submit | 语法 | public PromotionBargainVO submit  (String beginDate, String endDate) |
| 前置条件 | 特价包的商品、总价设置完成 |
| 后置条件 | 提交特价包 |
| PromotionBargian. buildPromotion | 语法 | private PromotionPO buildPromotion() |
| 前置条件 | 特价包的商品、总价设置完成 |
| 后置条件 | 建立特价包数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| PromotionDataService.  find(String ID) | 查找持久化数据 | |
| PromotionDataService.  insert(String condition) | 插入持久化数据 | |
| PromotionDataService.  delete(String condition) | 删除持久化对象 | |
| PromotionDataService.  update(String condition) | 更新持久化对象 | |

**PromotionClientController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionClientController.  show | 语法 | public ArrayList<PromotionClientVO> show() |
| 前置条件 | 创建一个PromotionClient对象 |
| 后置条件 | 调用PromotionClient的方法 |
| PromotionClientController. addGiftBill | 语法 | public void addGiftBill(InventoryBillVO giftBill) |
| 前置条件 | 创建一个PromotionClient对象 |
| 后置条件 | 调用PromotionClient的方法 |
| PromotionClientController. addGifts | 语法 | public void addGifts(CommodityItemVO vo) |
| 前置条件 | 创建一个PromotionClient对象 |
| 后置条件 | 调用PromotionClient的方法 |
| PromotionClientController.  setClient | 语法 | public voie setClient(ClientLevle level) |
| 前置条件 | 创建一个PromotionClient对象 |
| 后置条件 | 调用PromotionClient的方法 |
| PromotionClientController. submit | 语法 | public PromotionBargainVO submit  (PromoInputInfo info) |
| 前置条件 | 创建一个PromotionClient对象 |
| 后置条件 | 调用PromotionClient的方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| PromotionClient.show | 展示所有促销策略 | |
| PromotionClient. addGiftBill | 添加赠送单到赠品中 | |
| PromotionClient. addGifts | 添加赠品 | |
| PromotionClient.setClient | 设置客户等级 | |
| PromotionClient. submit | 提交促销策略 | |

**PromotionClient的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionClient.show | 语法 | public ArrayList<PromotionClientVO> show() |
| 前置条件 | 得到数据 |
| 后置条件 | 返回所有的特价包 |
| PromotionClient. addGiftBill | 语法 | public void addGiftBill(InventoryBillVO giftBill) |
| 前置条件 | 促销策略开始制定，并且输入合法 |
| 后置条件 | 添加赠送单到促销策略的赠品中 |
| PromotionClient. addGifts | 语法 | public void addGifts(CommodityItemVO vo) |
| 前置条件 | 促销策略开始制定，并且输入合法 |
| 后置条件 | 添加赠品到促销策略中 |
| PromotionClient.setClient | 语法 | public voie setClient(ClientLevle level) |
| 前置条件 | 促销策略开始制定，并且输入合法 |
| 后置条件 | 设置促销策略的客户等级 |
| PromotionClient. submit | 语法 | public PromotionClientVO submit  (PromoInputInfo info) |
| 前置条件 | 促销策略的赠品、客户等级设置完成 |
| 后置条件 | 提交促销策略 |
| PromotionClient. buildPromotion | 语法 | private PromotionPO buildPromotion() |
| 前置条件 | 促销策略的赠品、客户等级设置完成 |
| 后置条件 | 建立促销策略数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| PromotionDataService.  find(String ID) | 查找持久化数据 | |
| PromotionDataService.  insert(String condition) | 插入持久化数据 | |
| PromotionDataService.  delete(String condition) | 删除持久化对象 | |
| PromotionDataService.  update(String condition) | 更新持久化对象 | |

**PromotionTotalController的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionTotalController.  show | 语法 | public ArrayList<PromotionTotalVO> show() |
| 前置条件 | 创建一个PromotionTotal对象 |
| 后置条件 | 调用PromotionTotal的方法 |
| PromotionTotalController. addGiftBill | 语法 | public void addGiftBill(InventoryBillVO giftBill) |
| 前置条件 | 创建一个PromotionTotal对象 |
| 后置条件 | 调用PromotionTotal的方法 |
| PromotionTotalController. addGifts | 语法 | public void addGifts(CommodityItemVO vo) |
| 前置条件 | 创建一个PromotionTotal对象 |
| 后置条件 | 调用PromotionTotal的方法 |
| PromotionTotalController.  setTotal | 语法 | public voie setTotal(double total) |
| 前置条件 | 创建一个PromotionTotal对象 |
| 后置条件 | 调用PromotionTotal的方法 |
| PromotionTotalController. submit | 语法 | public PromotionTotalVO submit  (PromoInputInfo info) |
| 前置条件 | 创建一个PromotionTotal对象 |
| 后置条件 | 调用PromotionTotal的方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| PromotionClient.show | 展示所有促销策略 | |
| PromotionClient. addGiftBill | 添加赠送单到赠品中 | |
| PromotionClient. addGifts | 添加赠品 | |
| PromotionClient.setTotal | 设置总价 | |
| PromotionClient. submit | 提交促销策略 | |

**PromotionTotal的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionTotal.show | 语法 | public ArrayList<PromotionTotalVO> show() |
| 前置条件 | 得到数据 |
| 后置条件 | 返回所有的特价包 |
| PromotionTotal. addGiftBill | 语法 | public void addGiftBill(InventoryBillVO giftBill) |
| 前置条件 | 促销策略开始制定，并且输入合法 |
| 后置条件 | 添加赠送单到促销策略的赠品中 |
| PromotionTotal. addGifts | 语法 | public void addGifts(CommodityItemVO vo) |
| 前置条件 | 促销策略开始制定，并且输入合法 |
| 后置条件 | 添加赠品到促销策略中 |
| PromotionTotal.setTotal | 语法 | public voie setTotal(double total) |
| 前置条件 | 促销策略开始制定，并且输入合法 |
| 后置条件 | 设置促销策略的客户等级 |
| PromotionTotal. submit | 语法 | public PromotionTotalVO submit  (PromoInputInfo info) |
| 前置条件 | 促销策略的赠品、客户等级设置完成 |
| 后置条件 | 提交促销策略 |
| PromotionTotal. buildPromotion | 语法 | private PromotionPO buildPromotion() |
| 前置条件 | 促销策略的赠品、客户等级设置完成 |
| 后置条件 | 建立促销策略数据 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| PromotionDataService.  find(String ID) | 查找持久化数据 | |
| PromotionDataService.  insert(String condition) | 插入持久化数据 | |
| PromotionDataService.  delete(String condition) | 删除持久化对象 | |
| PromotionDataService.  update(String condition) | 更新持久化对象 | |

### Userbl模块

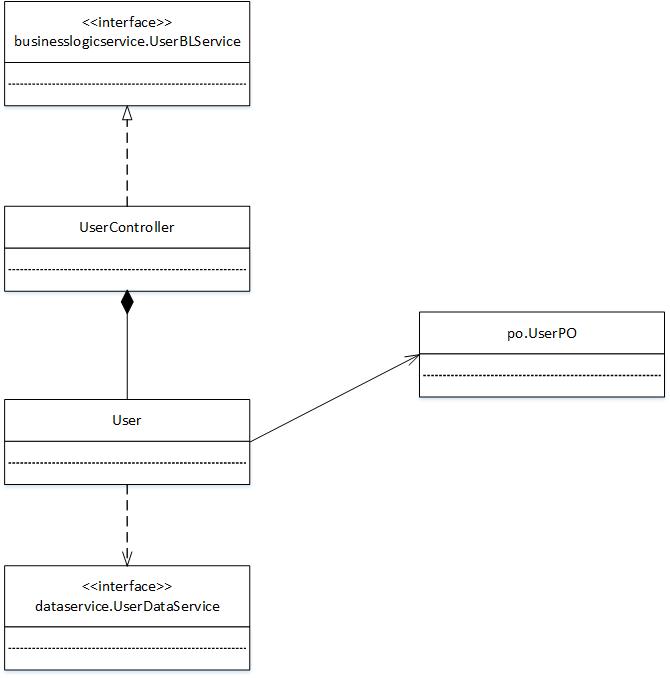
1. 模块概述

Userbl模块负责实现用户的管理具体功能需求和非功能需求可参见需求规格说明文档和体系结构设计文档。

1. 整体结构

根据体系结构的设计，我们将系统分为展示层、业务逻辑层、数据层。每一层之间为了增加灵活性，我们会添加接口。比如展示层和业务逻辑层之间，我们添加bussinesslogicservice.UserBLService接口。业务逻辑层和数据层之间添加dataservice.UserDataService接口。为了隔离业务逻辑职责和逻辑控制职责，我们增加了UserController，这样UserController会将对商品的逻辑业务处理委托给User对象。UserPO是作为客户的持久化对象被添加到设计模型中去的。

Userbl模块的设计如图



Userbl模块各个类的职责

|  |  |
| --- | --- |
| 模块 | 职责 |
| UserController | 负责实现用户管理界面所需要的服务 |
| User | 用户的领域模型对象，拥有用户的信息，可以帮助完成用户管理界面所需要的服务 |

1. 模块内类接口规范

**UserController的接口规范**

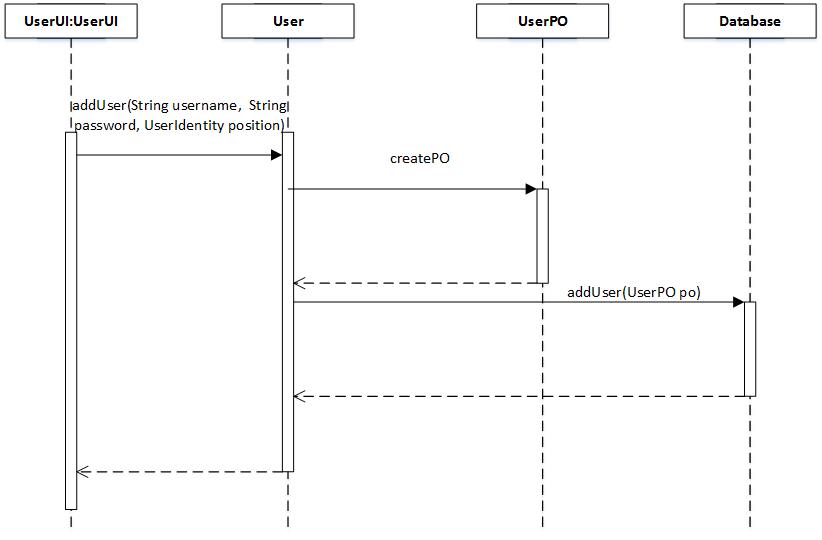
|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserController.login | 语法 | public ResultMessage login(String username, String password); |
| 前置条件 | 已创建一个User领域对象，password符合输入规则 |
| 后置条件 | 调用User领域的login方法 |
| UserController.addUser | 语法 | public ResultMessage addUser  (String username, String password, UserIdentity position) |
| 前置条件 | 已创建一个User领域对象，添加的User不重复 |
| 后置条件 | 调用User领域的addUser方法 |
| UserController.deletUser | 语法 | public ResultMessage deleteUser  (String username) |
| 前置条件 | 已创建一个User领域对象，id符合输入规则 |
| 后置条件 | 调用User领域的deleteUser方法 |
| UserController.updateUser | 语法 | public ResultMessage updateUser(String username, String password, UserIdentity newPosition, UserVO user) |
| 前置条件 | 已创建一个User领域对象，用户名、密码、职位符合输入规则 |
| 后置条件 | 调用User领域的updateUser方法 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| UserController.login | 用户登录 | |
| UserController.addUser | 添加一个用户 | |
| UserController.deletUser | 删除一个用户 | |
| UserController.updateUser | 更新一个用户 | |

**User接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| User.login | 语法 | public ResultMessage login(String username, String password); |
| 前置条件 | password符合输入规则 |
| 后置条件 | 查找是否存在相应的User，根据输入的password返回登录验证的结果 |
| User.addUser | 语法 | public ResultMessage addUser  (String username, String password, UserIdentity position) |
| 前置条件 | 添加的User不重复 |
| 后置条件 | 添加User |
| User.deletUser | 语法 | public ResultMessage deleteUser  (String username) |
| 前置条件 | id符合输入规则 |
| 后置条件 | 查找相应的User，根据输入的id删除该用户 |
| User.updateUser | 语法 | public ResultMessage updateUser(String username, String password, UserIdentity newPosition, UserVO user) |
| 前置条件 | 用户名、密码、职位符合输入规则 |
| 后置条件 | 查找相应的User，根据输入的id，更改该用户的权限 |
| 需要的服务（需接口） | | |
| 服务名 | 服务 | |
| DatabaseFacory.getUserDataBase | 得到User的服务的引用 | |
| UserDateService.find  (UserPO po) | 查找单一持久化对象 | |
| UserDataService.insert  (UserPO po) | 插入单一持久化对象 | |
| UserDataService.delete  (UserPO po) | 删除单一持久化对象 | |
| userDataService.update  (UserPO po) | 更新单一持久化对象 | |

1. 业务逻辑层的动态模型

下图表明了灯具管理系统中，添加一个用户时，用户业务逻辑处理的相关对象之间的协作。然后是添加用户的状态图



添加用户顺序图



User状态图

1. 用户逻辑层的设计原理

利用委托式的控制风格，每个界面需要访问的业务逻辑由各自的控制器委托给不同的领域对象。

## 数据层的分解

数据层的开发包图请参见体系结构描述文档图4。

### DataFactoryService

1. 模块概述

DataFactoryService模块是一个工厂类，用来创建其他的DataService实例，并将其统一管理起来。

1. 整体结构

DataFactoryService模块设计如下图所示



1. 模块内部的接口规范

DataFactory内部的的接口规范如下表

**DataFactory的接口规范**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 提供的服务（供接口） | | | | | |
| DataFactory.getAccountaInitData | | 语法 | | public DataService getAccountaInitData() throws RemoteException | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回AccountaInitData对象 | |
| DataFactory. getAccountData | | 语法 | | public DataService getAccountData() throws RemoteException; | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回AccountData对象 | |
| DataFactory. getApprovalData | | 语法 | | public DataService getApprovalData() throws RemoteException; | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回ApprovalData对象 | |
| DataFactory. getClientData | | 语法 | | public DataService getClientData() throws RemoteException; | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回ClientData对象 | |
| DataFactory. getCommodityData | | 语法 | | public DataService getCommodityData() throws RemoteException; | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回CommodityData对象 | |
| DataFactory.  getCommoditySortData | | 语法 | | public DataService getCommoditySortData () throws RemoteException; | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回CommoditySortData对象 | |
| DataFactory. getAccountBillData | | 语法 | | public DataService getAccountBillData () throws RemoteException | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回AccountBillData对象 | |
| DataFactory. getInventoryData | | 语法 | | public DataService getInventoryData () throws RemoteException | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回InventoryData对象 | |
| DataFactory. getPromotionData | | 语法 | | public DataService getPromotionData () throws RemoteException | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回PromotionData对象 | |
| DataFactory. getPurchaseData | | 语法 | | public DataService getPurchaseData () throws RemoteException | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回PurchaseData对象 | |
| DataFactory. getRecordData | | 语法 | | public DataService getRecordData () throws RemoteException | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回RecordData对象 | |
| DataFactory. getSaleData | | 语法 | | public DataService getSaleData () throws RemoteException | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回SaleData对象 | |
| DataFactory. getUserData | | 语法 | | public DataService getUserData () throws RemoteException | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回UserData对象 | |
| DataFactory. getCashBillData | | 语法 | | public DataService getCashBillData () throws RemoteException | |
| 前置条件 | | 无 | |
| 后置条件 | | 返回CashBillData对象 | |

### InventoryDataService

1. 模块概述

InventoryDataService模块提供库存单据的提交服务（赠送单、报溢单、报损单、报警单）

1. 模块内类接口规范

InventoryDataService接口规范如下表

**InventoryDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| InventoryDataService.insert | 语法 | public ResultMessage insert(InventoryBillPO po) |
| 前置条件 | 无 |
| 后置条件 | 返回提交是否成功 |
| InventoryDataService.find | 语法 | public InventoryBillPO find(String ID) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| InventoryDataService.update | 语法 | public ResultMessage update (InventoryBillPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| InventoryDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService.  getGiftID | 语法 | public String getGiftID () |
| 前置条件 | 无 |
| 后置条件 | 得到赠送单ID |
| InventoryDataService.  getLossID | 语法 | public String getLossID () |
| 前置条件 | 无 |
| 后置条件 | 得到报损单ID |
| InventoryDataService.  getOverflowID | 语法 | public String getOverflowID () |
| 前置条件 | 无 |
| 后置条件 | 得到报溢单ID |
| InventoryDataService.  getAlarmID | 语法 | public String getAlarmID () |
| 前置条件 | 无 |
| 后置条件 | 得到报警单ID |
| InventoryDataService.  show | 语法 | public ArrayList<InventoryPO> show() |
| 前置条件 | 无 |
| 后置条件 | 返回所有库存单据的数据 |

### PurchaseDataService

1. 模块概述

PurchaseDataService提供进货（进货退货）数据集体载入、保存、增加、删除、查找

1. 模块内部类的接口规范

PurchaseDataService类的接口规范如下表所示

**PurchaseDataService类接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PurchaseDataService.insert | 语法 | public ResultMessage insert(PurchasePO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| PurchaseDataService.find | 语法 | public PurchasePO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| PurchaseDataService.delete | 语法 | public ResultMessage delete (String ID) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| PurchaseDataService.update | 语法 | public ResultMessage update (PurchasePO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| PurchaseDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| PurchaseDataService.  getPurchaseID | 语法 | public String getPurchaseID () |
| 前置条件 | 无 |
| 后置条件 | 得到进货单ID |
| PurchaseDataService.  getPurchaseBackID | 语法 | public String getPurchaseBackID () |
| 前置条件 | 无 |
| 后置条件 | 得到进货退货单ID |
| PurchaseDataService.show | 语法 | public ArrayList<PurchasePO> show() |
| 前置条件 | 无 |
| 后置条件 | 得到所有进货、进货退货单数据 |

### AccountBillDataService

1. 模块概述

AccountBillDataService提供收款单、付款单、现金单的集体载入、保存、查看修改操作

1. 模块内类的接口规范

AccountBillDataService类的接口规范如下表所示

**AccountBillDataService类的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountBillDataService.  insert | 语法 | public ResultMessage insert  (AccountBillPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| AccountBillDataService.find | 语法 | public AccountBillPO find(String ID) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| AccountBillDataService.  delete | 语法 | public ResultMessage delete  (String ID) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| AccountBillDataService.  update | 语法 | public ResultMessage update  (AccountBillPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| AccountBillDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| AccountBillDataService. getPayID | 语法 | public String getPayID () |
| 前置条件 | 无 |
| 后置条件 | 返回付款单ID |
| AccountBillDataService. getExpenseID | 语法 | public String getExpenseID () |
| 前置条件 | 无 |
| 后置条件 | 返回收款单ID |
| AccountBillDataService.show | 语法 | public ArrayList<AccountBillPO> show() |
| 前置条件 | 无 |
| 后置条件 | 返回所有收款单、付款单 |

### ApprovalDataService

1. 模块概述

ApproalDataService提供需要审批的单据的集体载入、保存、查看修改操作

1. 模块内部类的接口

**ApproalDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ApprovalDataService.update | 语法 | public ResultMessage update(PersistentObject po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| ApprovalDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### UserDataService

1. 模块概述

提供用户集体载入、保存、增删改查服务

1. 模块内部类的接口

**UserDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| UserDataService.insert | 语法 | public ResultMessage insert(UserPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| UserDataService.find | 语法 | public UserPO find(String username) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| UserDataService.delete | 语法 | public ResultMessage delete (String ID) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| UserDataService.update | 语法 | public ResultMessage update (UserPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| UserDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### CommodityDataService

1. 模块概述

提供商品集体载入、保存、增删改查服务

1. 模块内部类的接口

**CommodityDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommodityDataService.insert | 语法 | public ResultMessage insert(CommodityPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| CommodityDataService.find | 语法 | public ResultMessage find(String name) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| CommodityDataService.delete | 语法 | public ResultMessage delete (String ID) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| CommodityDataService.update | 语法 | public ResultMessage update (CommodityPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| CommodityDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### CommoditySortDataService

1. 模块概述

提供商品分类数据集体载入、保存、增删改查服务

1. 模块内部类的接口

**CommoditySortDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| CommoditySortDataService.insert | 语法 | public ResultMessage insert  (CommoditySortPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| CommoditySortDataService.find | 语法 | public CommoditySortPO find(String name) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| CommoditySortDataService.delete | 语法 | public ResultMessage delete  (String id) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| CommoditySortDataService.update | 语法 | public ResultMessage update  (CommoditySortPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| CommoditySortDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| CommoditySortDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| CommoditySortDataService. show | 语法 | public ArrayList<CommoditySortPO>  show () |
| 前置条件 | 无 |
| 后置条件 | 返回所有商品分类 |

### SaleDataService

1. 模块概述

提供销售（销售退货）数据集体载入、保存、增加、删除、查找

1. 模块内部类的接口

**SaleDataService的接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| SaleDataService.insert | 语法 | public ResultMessage insert(SalesPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| SaleDataService.find | 语法 | public SalesPO find(String id) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| SaleDataService.delete | 语法 | public ResultMessage delete (String id) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| SaleDataService.update | 语法 | public ResultMessage update (SalesPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| SaleDataService. show | 语法 | public ArrayList<SalesPO>show () |
| 前置条件 | 无 |
| 后置条件 | 返回所有销售单 |
| SaleDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getSaleID | 语法 | public String getSaleID () |
| 前置条件 | 无 |
| 后置条件 | 返回销售单单据ID |
| InventoryDataService. getSaleBackID | 语法 | public String getSaleBackID () |
| 前置条件 | 无 |
| 后置条件 | 返回销售退货单单据ID |

### AccountDataService

1. 模块概述

提供账户数据集体载入、保存、增删改查服务

1. 模块内部类的接口

**AccountDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountDataService.insert | 语法 | public ResultMessage insert(AccountPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| AccountDataService.find | 语法 | public AccountPO find(String name) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| AccountDataService.delete | 语法 | public ResultMessage delete (String name) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| AccountDataService.update | 语法 | public ResultMessage update (AccountPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| AccountDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### ClientDataService

1. 模块概述

提供客户数据集体载入、保存、增删改查服务

1. 模块内部类的接口

**ClientDataService类接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| ClientDataService.insert | 语法 | public ResultMessage insert(ClientPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| ClientDataService.find | 语法 | public ClientPO find(String name) |
| 前置条件 | 无 |
| 后置条件 | 查找返回相应的结果 |
| ClientDataService.delete | 语法 | public ResultMessage delete (String id) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 删除po |
| ClientDataService.update | 语法 | public ResultMessage update (ClientPO po) |
| 前置条件 | 同样的po在数据中存在 |
| 后置条件 | 更新po |
| ClientDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### AccountaInitDataService

1. 模块概述

提供期初建账数据集体载入、保存、增加、查看服务

1. 模块内部类的接口

**AccountaInitDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| AccountaInitDataService.insert | 语法 | public ResultMessage insert(AccountaInitPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| AccountaInitDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |
| AccountaInitDataService.getAllMsg | 语法 | public ArrayList<AccountaInitPO> getAllMsg (); |
| 前置条件 | 无 |
| 后置条件 | 返回全部AccountaInitPO持久数据 |
| InventoryDataService. getID | 语法 | public void getID () |
| 前置条件 | 无 |
| 后置条件 | 初始化持久数据 |

### PromotionDataService

1. 模块概述

提供制定促销策略的数据集体载入、保存、增加、查看服务

1. 模块内部类的接口

**PromotionDataService接口规范**

|  |  |  |
| --- | --- | --- |
| 提供的服务（供接口） | | |
| PromotionDataService.getID | 语法 | public String getID() |
| 前置条件 | 无 |
| 后置条件 | 返回ID |
| PromotionDataService.init | 语法 | public void init() |
| 前置条件 | 无 |
| 后置条件 | 初始化数据 |
| PromotionDataService.find | 语法 | public PO find(String ID) |
| 前置条件 | 无 |
| 后置条件 | 返回根据ID查到的数据 |
| PromotionDataService.insert | 语法 | public ResultMessage insert(PromotionPO po) |
| 前置条件 | 同样的po在数据中不存在 |
| 后置条件 | 增加一条po数据 |
| PromotionDataService.update | 语法 | public void update(PromotionPO po) |
| 前置条件 | 无 |
| 后置条件 | 更改该po数据 |
| PromotionDataService.delete | 语法 | public ArrayList<AccountaInitPO>  delete(String ID) |
| 前置条件 | 无 |
| 后置条件 | 根据ID删除促销策略 |
| PromotionDataService. show | 语法 | public ArrayList<PromotionPO>show () |
| 前置条件 | 无 |
| 后置条件 | 返回所有促销策略 |
| PromotionDataService.show | 语法 | public ArrayList<PromotionPO> show  (PromotionType type) |
| 前置条件 | 无 |
| 后置条件 | 根据类型返回特定的促销策略 |

# 依赖视角

下图是客户端和服务器端各自的包之间的依赖关系

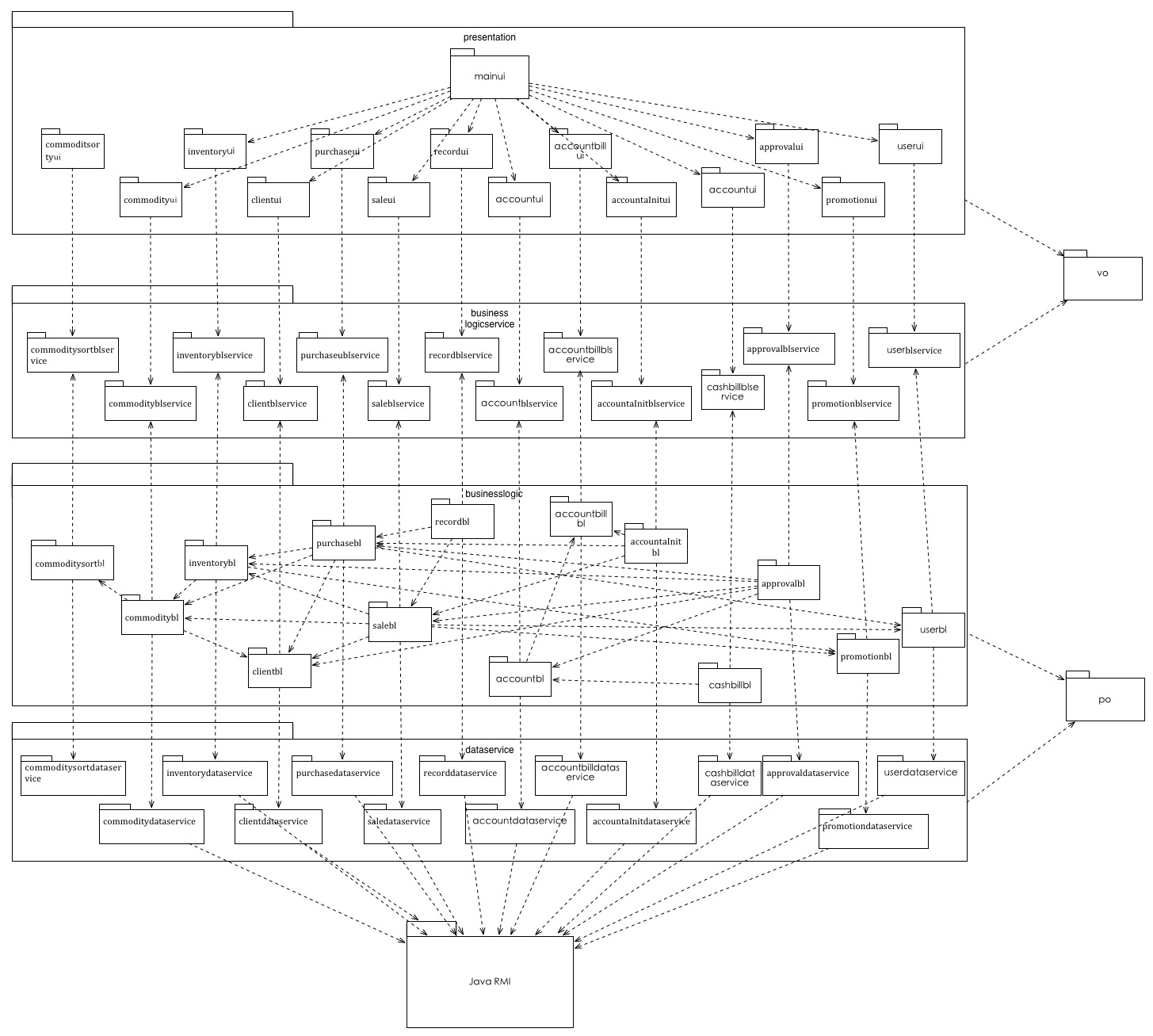


图 1 客户端包图

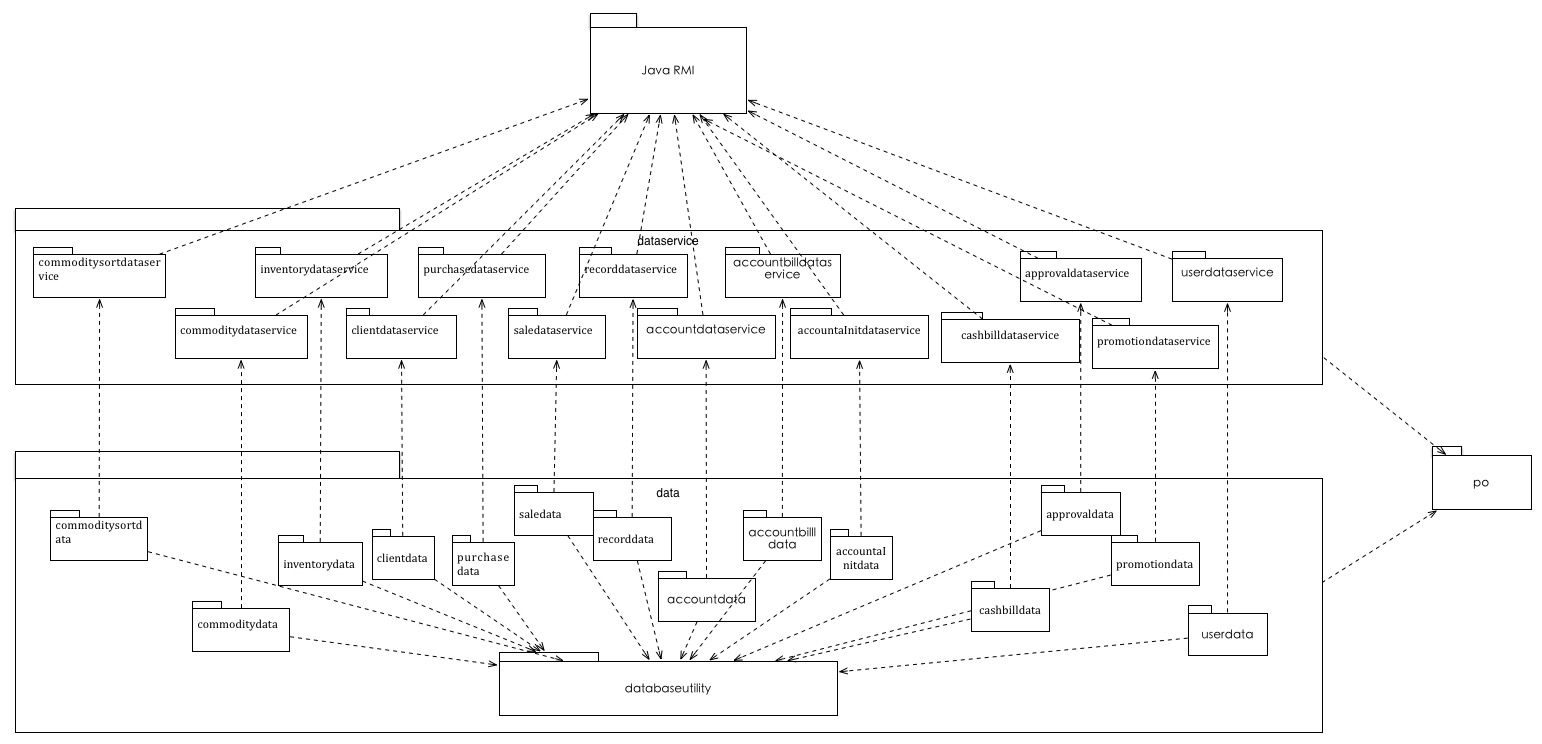


图 2 服务器端包图